Javier Esparza
Andrzej S. Murawski (Eds.)

LNCS 10203 ARCoSS

Foundations of Software Science and Computation Structures

20th International Conference, FOSSACS 2017 Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017 Uppsala, Sweden, April 22–29, 2017, Proceedings

Springer

Lecture Notes in Computer Science **10203**

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

Advanced Research in Computing and Software Science
Subline of Lecture Notes in Computer Science

More information about this series at http://www.springer.com/series/7407

Javier Esparza · Andrzej S. Murawski (Eds.)

Foundations of Software Science and Computation Structures

20th International Conference, FOSSACS 2017
Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017
Uppsala, Sweden, April 22–29, 2017
Proceedings

Springer

Editors
Javier Esparza
TU München
Garching, Bayern
Germany

Andrzej S. Murawski
University of Warwick
Coventry
UK

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-662-54457-0 ISBN 978-3-662-54458-7 (eBook)
DOI 10.1007/978-3-662-54458-7

Library of Congress Control Number: 2017933275

LNCS Sublibrary: SL1 – Theoretical Computer Science and General Issues

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer-Verlag GmbH Germany
The registered company address is: Heidelberger Platz 3, 14197 Berlin, Germany

ETAPS Foreword

Welcome to the proceedings of ETAPS 2017, which was held in Uppsala! It was the first time ever that ETAPS took place in Scandinavia.

ETAPS 2017 was the 20th instance of the European Joint Conferences on Theory and Practice of Software. ETAPS is an annual federated conference established in 1998, and consists of five conferences: ESOP, FASE, FoSSaCS, TACAS, and POST. Each conference has its own Program Committee (PC) and its own Steering Committee. The conferences cover various aspects of software systems, ranging from theoretical computer science to foundations to programming language developments, analysis tools, formal approaches to software engineering, and security. Organizing these conferences in a coherent, highly synchronized conference program enables participation in an exciting event, offering the possibility to meet many researchers working in different directions in the field and to easily attend talks of different conferences. Before and after the main conference, numerous satellite workshops take place and attract many researchers from all over the globe.

ETAPS 2017 received 531 submissions in total, 159 of which were accepted, yielding an overall acceptance rate of 30%. I thank all authors for their interest in ETAPS, all reviewers for their peer reviewing efforts, the PC members for their contributions, and in particular the PC (co-)chairs for their hard work in running this entire intensive process. Last but not least, my congratulations to all authors of the accepted papers!

ETAPS 2017 was enriched by the unifying invited speakers Kim G. Larsen (Aalborg University, Denmark) and Michael Ernst (University of Washington, USA), as well as the conference-specific invited speakers (FoSSaCS) Joel Ouaknine (MPI-SWS, Germany, and University of Oxford, UK) and (TACAS) Dino Distefano (Facebook and Queen Mary University of London, UK). In addition, ETAPS 2017 featured a public lecture by Serge Abiteboul (Inria and ENS Cachan, France). Invited tutorials were offered by Véronique Cortier (CNRS research director at Loria, Nancy, France) on security and Ken McMillan (Microsoft Research Redmond, USA) on compositional testing. My sincere thanks to all these speakers for their inspiring and interesting talks!

ETAPS 2017 took place in Uppsala, Sweden, and was organized by the Department of Information Technology of Uppsala University. It was further supported by the following associations and societies: ETAPS e.V., EATCS (European Association for Theoretical Computer Science), EAPLS (European Association for Programming Languages and Systems), and EASST (European Association of Software Science and Technology). Facebook, Microsoft, Amazon, and the city of Uppsala financially supported ETAPS 2017. The local organization team consisted of Parosh Aziz Abdulla (general chair), Wang Yi, Björn Victor, Konstantinos Sagonas, Mohamed Faouzi Atig, Andreina Francisco, Kaj Lampka, Tjark Weber, Yunyun Zhu, and Philipp Rümmer.

The overall planning for ETAPS is the main responsibility of the Steering Committee, and in particular of its executive board. The ETAPS Steering Committee

consists of an executive board, and representatives of the individual ETAPS conferences, as well as representatives of EATCS, EAPLS, and EASST. The executive board consists of Gilles Barthe (Madrid), Holger Hermanns (Saarbrücken), Joost-Pieter Katoen (chair, Aachen and Twente), Gerald Lüttgen (Bamberg), Vladimiro Sassone (Southampton), Tarmo Uustalu (Tallinn), and Lenore Zuck (Chicago). Other members of the Steering Committee are: Parosh Abdulla (Uppsala), Amal Ahmed (Boston), Christel Baier (Dresden), David Basin (Zurich), Lujo Bauer (Pittsburgh), Dirk Beyer (Munich), Giuseppe Castagna (Paris), Tom Crick (Cardiff), Javier Esparza (Munich), Jan Friso Groote (Eindhoven), Jurriaan Hage (Utrecht), Reiko Heckel (Leicester), Marieke Huisman (Twente), Panagotios Katsaros (Thessaloniki), Ralf Küsters (Trier), Ugo del Lago (Bologna), Kim G. Larsen (Aalborg), Axel Legay (Rennes), Matteo Maffei (Saarbrücken), Tiziana Margaria (Limerick), Andrzej Murawski (Warwick), Catuscia Palamidessi (Palaiseau), Julia Rubin (Vancouver), Alessandra Russo (London), Mark Ryan (Birmingham), Don Sannella (Edinburgh), Andy Schürr (Darmstadt), Gabriele Taentzer (Marburg), Igor Walukiewicz (Bordeaux), and Hongseok Yang (Oxford).

I would like to take this opportunity to thank all speakers, attendees, organizers of the satellite workshops, and Springer for their support. Finally, a big thanks to Parosh and his local organization team for all their enormous efforts enabling a fantastic ETAPS in Uppsala!

January 2017 Joost-Pieter Katoen

Preface

This volume contains the papers presented at the 20th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2017), which was held April 24–27, 2017, in Uppsala, Sweden. The conference is dedicated to foundational research with a clear significance for software science and brings together research on theories and methods to support the analysis, integration, synthesis, transformation, and verification of programs and software systems.

In addition to an invited talk by Joël Ouaknine (MPI-SWS Saarbrücken and University of Oxford) on "Fundamental Algorithmic Problems and Challenges in Dynamical and Cyber-Physical Systems," the program consisted of 32 contributed papers, selected from among 101 submissions. Each submission was assessed by three or more Program Committee members. The conference management system EasyChair was used to handle the submissions, to conduct the electronic Program Committee discussions, and to assist with the assembly of the proceedings.

We wish to thank all the authors who submitted papers for consideration, the members of the Program Committee for their conscientious work, and all additional reviewers who assisted the Program Committee in the evaluation process. Finally, we would like to thank the ETAPS organization for providing an excellent environment for FoSSaCS, other conferences, and workshops.

January 2017

Javier Esparza
Andrzej Murawski

Organization

Program Committee

Mohamed Faouzi Atig	Uppsala University, Sweden
Jos Baeten	CWI (Centrum Wiskunde & Informatica), The Netherlands
Christel Baier	Technical University of Dresden, Germany
Filippo Bonchi	University of Pisa, Italy
Tomáš Brázdil	Masaryk University, Czech Republic
James Brotherston	University College London, UK
Anuj Dawar	University of Cambridge, UK
Michael Emmi	Bell Labs, Nokia
Javier Esparza	Technische Universität München, Germany
Rajeev Gore	The Australian National University, Australia
Stefan Göller	LSV, CNRS, ENS Cachan, France
Thomas Hildebrandt	IT University of Copenhagen, Denmark
Delia Kesner	Université Paris-Diderot, France
Sławomir Lasota	Warsaw University, Poland
Anthony Widjaja Lin	University of Oxford, UK
Roland Meyer	TU Braunschweig, Germany
Aniello Murano	Università degli Studi di Napoli Federico II, Italy
Andrzej Murawski	University of Warwick, UK
Simona Ronchi Della Rocca	Università di Torino, Italy
Jan Rutten	CWI, The Netherlands
Margus Veanes	Microsoft Research
Lijun Zhang	Institute of Software, Chinese Academy of Sciences, China

Additional Reviewers

Accattoli, Beniamino	Basold, Henning
Aceto, Luca	Belardinelli, Francesco
Adamek, Jiri	Benes, Nikola
Aiswarya, C.	Bertrand, Nathalie
Altenkirch, Thorsten	Berwanger, Dietmar
Ancona, Davide	Bieniusa, Annette
Angiuli, Carlo	Birkedal, Lars
Asarin, Eugene	Bizjak, Aleš
Bacci, Giorgio	Bonelli, Eduardo
Bacci, Giovanni	Bonfante, Guillaume
Baelde, David	Borgström, Johannes

Bozzelli, Laura
Bradfield, Julian
Brenguier, Romain
Brihaye, Thomas
Brunet, Paul
Carbone, Marco
Cardone, Felice
Castellan, Simon
Cerone, Andrea
Chaudhuri, Kaustuv
Chini, Peter
Clouston, Ranald
Colcombet, Thomas
Cosme Llópez, Enric
Crole, Roy
D'Antoni, Loris
D'Osualdo, Emanuele
Dal Lago, Ugo
Danos, Vincent
Dawson, Jeremy
Debois, Søren
Della Monica, Dario
Demangeon, Romain
Docherty, Simon
Dragoi, Cezara
Ehrhard, Thomas
Enea, Constantin
Escardo, Martin
Faggian, Claudia
Fijalkow, Nathanaël
Forejt, Vojtech
Fu, Hongfei
Galmiche, Didier
Ganty, Pierre
Garnier, Ilias
Gburek, Daniel
Ghica, Dan
Ghilezan, Silvia
Gimbert, Hugo
Gimenez, Stéphane
Goncharov, Sergey
Goubault-Larrecq, Jean
Grellois, Charles
Groote, Jan Friso
Grygiel, Katarzyna
Haase, Christoph
Hahn, Ernst Moritz
Hirschowitz, Tom
Hofman, Piotr
Hofstra, Pieter
Holik, Lukas
Horn, Florian
Horne, Ross
Hou, Zhe
Hsu, Justin
Hunter, Paul
Jaber, Guilhem
Jacobs, Bart
Jansen, David
Jansen, David N.
Jung, Jean Christoph
Kernberger, Daniel
Kiefer, Stefan
Kissinger, Aleks
Klin, Bartek
Koslowski, Jürgen
Krebbers, Robbert
Kretinsky, Jan
Kucera, Antonin
Kumar, Ramana
Kupke, Clemens
König, Barbara
Laird, James
Le, Quang Loc
Lee, Matias David
Lellmann, Bjoern
Leroux, Jérôme
Lescanne, Pierre
Licata, Daniel R.
Lozes, Etienne
Lumsdaine, Peter Lefanu
Luttik, Bas
Malvone, Vadim
Manuel, Amaldev
Manzonetto, Giulio
Mardare, Radu
Martens, Wim
Maubert, Bastien
Mazza, Damiano
McCusker, Guy
Melliès, Paul-André
Merro, Massimo

Milius, Stefan
Mimram, Samuel
Mio, Matteo
Mousavi, Mohammadreza
Munoz Fuentes, Pablo
Muskalla, Sebastian
Møgelberg, Rasmus Ejlers
Novotný, Petr
Obdrzalek, Jan
Ong, Luke
Otop, Jan
Ouaknine, Joel
Panangaden, Prakash
Patrizi, Fabio
Pattinson, Dirk
Perelli, Giuseppe
Peron, Adriano
Perrin, Matthieu
Peters, Kirstin
Petrisan, Daniela
Petrov, Tatjana
Pitts, Andrew
Popescu, Andrei
Power, John
Pradic, Pierre
Quaas, Karin
Regis-Gianas, Yann
Rehak, Vojtech
Reniers, Michel
Rosa-Velardo, Fernando
Rot, Jurriaan
Roversi, Luca
Rowe, Reuben
Rubin, Sasha
S., Krishna
Saarikivi, Olli
Saivasan, Prakash
Salamanca, Julian
Sammartino, Matteo
Sauro, Luigi
Scherer, Gabriel
Schubert, Aleksy
Schuermann, Carsten
Schöpp, Ulrich
Scott, Phil
Seiller, Thomas
Skrzypczak, Michał
Sojakova, Kristina
Sokolova, Ana
Srba, Jiri
Srivathsan, B.
Staton, Sam
Stratulat, Sorin
Streicher, Thomas
Swamy, Nikhil
Tabareau, Nicolas
Tan, Tony
Toruńczyk, Szymon
Tsukada, Takeshi
Turrini, Andrea
Tzevelekos, Nikos
Urbat, Henning
Uustalu, Tarmo
van de Pol, Jaco
Van Raamsdonk, Femke
Veltri, Niccolò
Vortmeier, Nils
Wiedijk, Freek
Winter, Joost
Woltzenlogel Paleo, Bruno
Worrell, James
Wu, Zhilin
Zaionc, Marek
Zanasi, Fabio
Zetzsche, Georg
Zorzi, Margherita
Zunino, Roberto

Fundamental Algorithmic Problems and Challenges in Dynamical and Cyber-Physical Systems (Abstract of Invited Talk)

Joël Ouaknine[1, 2]

[1] Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany
[2] Department of Computer Science, Oxford University, Oxford, UK

Abstract. Dynamical and cyber-physical systems, whose continuous evolution is subject to differential equations, permeate vast areas of engineering, physics, mathematics, and computer science. In this talk, I consider a selection of fundamental algorithmic problems for such systems, such as reachability, invariant synthesis, and controllability. Although the decidability and complexity of many of these problems are open, some partial and conditional results are known, occasionally resting on certain number-theoretic hypotheses such as Schanuel's conjecture. More generally, the study of algorithmic problems for dynamical and cyber-physical systems draws from an eclectic array of mathematical tools, ranging from Diophantine approximation to algebraic geometry. I will present a personal and select overview of the field and discuss areas of current active research.

Contents

Automata, Logic and Formal Languages

Proof Theory

Probability

Concurrency

Lambda Calculus and Constructive Proof

Semantics and Category Theory

Coherence Spaces and Higher-Order Computation

Coherence Spaces and Uniform Continuity

Kei Matsumoto(✉)

RIMS, Kyoto University, Kyoto, Japan
kmtmt@kurims.kyoto-u.ac.jp

Abstract. We consider a model of classical linear logic based on coherence spaces endowed with a notion of totality. If we restrict ourselves to total objects, each coherence space can be regarded as a uniform space and each linear map as a uniformly continuous function. The linear exponential comonad then assigns to each uniform space $\boldsymbol{X}$ the finest uniform space $!\boldsymbol{X}$ compatible with $\boldsymbol{X}$. By a standard realizability construction, it is possible to consider a theory of representations in our model. Each (separable, metrizable) uniform space, such as the real line $\mathbb{R}$, can then be represented by (a partial surjecive map from) a coherence space with totality. The following holds under certain mild conditions: a function between uniform spaces $\mathbb{X}$ and $\mathbb{Y}$ is uniformly continuous if and only if it is realized by a total linear map between the coherence spaces representing $\mathbb{X}$ and $\mathbb{Y}$.

1 Introduction

Since the inception of Scott's domain theory in 1960's, *topology* and *continuity* have been playing a prominent role in denotational understanding of logic and computation. On the other hand, *uniformity* and *uniform continuity* have not yet been explored so much. The purpose of this paper is to bring them into the setting of denotational semantics by relating them to another denotational model: *coherence spaces* and *linear maps*. Our principal idea is that linear maps should be uniformly continuous, not just in analysis, but also in denotational semantics. The following situation, typical for computable real functions (in the sense of [Ko91]), illustrates our idea.

Example 1. Imagine that each real number $x \in \mathbb{R}$ is presented by a rational Cauchy sequence $(x_n)_{n\in\mathbb{N}}$ with $|x - x_n| \leq 2^{-n}$. Let $f : \mathbb{R} \to \mathbb{R}$ be a computable function which is uniformly continuous. Then there must be a function $\mu : \mathbb{N} \to \mathbb{N}$, called a *modulus of continuity*, such that an approximation of $f(x)$ with precision 2^{-m} can be computed from a single rational number $x_{\mu(m)}$, no matter where x is located on the real line. Thus one has to access the sequence (x_n) (regarded as an oracle) only once.

On the other hand, if $f : \mathbb{R} \to \mathbb{R}$ is not uniformly continuous, it admits no uniform modulus of continuity. Hence one has to access (x_n) at least twice to obtain an approximation of $f(x)$, once for figuring out the location of x and thus obtaining a local modulus of continuity μ around x, and once for getting the approximate value $x_{\mu(m)}$.

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 3–19, 2017.
DOI: 10.1007/978-3-662-54458-7_1

Thus there is a difference in *query complexity* between uniformly continuous and non-uniformly continuous functions. This observation leads us to an inspiration that linear maps, whose query complexity is 1, should be somehow related to uniformly continuous functions. To formalize this intuition, we work with coherence spaces with totality.

Coherence spaces, introduced by Girard [Gi87], are Scott-Ershov domains which are simply presented as undirected reflexive graphs. It was originally introduced as a denotational semantics for System F, and later led to the discovery of linear logic. One of the notable features of coherence spaces is that there are two kinds of morphisms coexisting: stable and linear maps.

Totalities, which originate in domain theory (eg. [Gi86, No90, Be93]), are often attached to coherence spaces (eg. [KN97]). Specifically, a *totality* on a coherence space $\boldsymbol{X}$ in our sense is a set of cliques $\mathcal{T}_{\boldsymbol{X}}$ which is equivalent to its bi-orthogonal $\mathcal{T}_{\boldsymbol{X}}^{\perp\perp}$ with respect to the orthogonal relation defined by $a \perp \mathfrak{c} \Leftrightarrow a \cap \mathfrak{c} \neq \varnothing$ for $a \in \boldsymbol{X}$ and $\mathfrak{c} \in \boldsymbol{X}^{\perp}$. Totalities are usually employed to restrict objects and morphisms to total ones to interpret well-behaved programs, while we emphasize the role of *co-totalities* as uniform structures on totalities. When restricted to "strict" ones (to be defined later), a totality $\mathcal{T}_{\boldsymbol{X}}$ can be seen as a set of *ideal points* of a uniform space $\mathbb{X}$, while the co-totality $\mathcal{T}_{\boldsymbol{X}}^{\perp}$ as a *uniform sub-basis* for $\mathbb{X}$. Moreover, this allows us to prove that every "total" linear map $F : \boldsymbol{X} \longrightarrow_{lin} \boldsymbol{Y}$ is uniformly continuous (though not vice versa).

The category of coherence spaces with totality and total linear maps is a model of classical linear logic, by simply extending the constructions on coherence spaces to totalities. In this setting, the linear exponential comonad ! admits an interesting interpretation: it assigns to each uniform space $\mathbb{X}$ the *finest* uniform space compatible with $\mathbb{X}$. Curiously, this phenomenon seems very specific to the coherence spaces model among various models of linear logic. Especially, the "internal completeness" property (Proposition 1) highly depends on the graph structure of coherence spaces.

We then apply our framework to *computable analysis*, where people study computation over various continuous and analytic structures (such as Euclidean spaces, metric spaces and Banach spaces). An essential prerequisite for arguing computability over such structures is that each space should be *concretely* represented. Traditional approaches use partial surjective maps (*representations*) from concrete structures such as the Baire space [KW85, We00, BHW08] or Scott-Ershov domains [Bl97, ES99, SHT08]. Instead of them, we consider partial surjective maps from coherence spaces (with totality).

This program has been already launched by [MT16], where we have suitably defined *admissible* representations based on coherence spaces (by importing various results from the type-two theory of effectivity [We00]). The principal result there is as follows. Let $\mathbb{X}$ and $\mathbb{Y}$ be topological spaces admissibly represented by coherence spaces $\boldsymbol{X}$ and $\boldsymbol{Y}$ (eg. the real line $\mathbb{R}$ is admissibly represented by a coherence space $\boldsymbol{R}$ in Example 2). Then a function $f : \mathbb{X} \to \mathbb{Y}$ is realized (i.e., tracked) by a stable map $F : \boldsymbol{X} \longrightarrow_{st} \boldsymbol{Y}$ if and only if it is *sequentially continuous*: it preserves the limit of any convergent sequence in $\mathbb{X}$.

In [MT16], we also have found a linear variant of the above correspondence: when restricted to real functions, f is realized by a *linear* map $F : \boldsymbol{R} \longrightarrow_{lin} \boldsymbol{R}$ if and only if it is *uniformly continuous*. Thus linearity indeed corresponds to uniform continuity, at least, for real functions. While we did not have any rationale or generalization at that time, we now have a better understanding of uniform continuity in terms of coherence spaces with totality. We are now able to generalize the above result on $\mathbb{R}$ to arbitrary chain-connected separable metrizable uniform spaces as a consequence of the fact that *total linear maps are uniformly continuous*.

Due to lack of space, some proofs are omitted and can be found in the full version which will be available at arxiv.org.

Plan of the paper. We quickly review uniform spaces in Sect. 2.1 and coherence spaces in Sect. 2.2. We then introduce in Sect. 3.1 the notion of (co)totality and strictness of (anti-)cliques, which naturally extend the structure of coherence spaces. In Sect. 3.2, we explore two uniformities induced by co-totalities and the role of the linear exponential comonad ! in this interpretation. In Sect. 4, we give an application of our model to computable analysis. We conclude in Sect. 5 with some future work.

2 Preliminaries

2.1 Uniform Spaces

We review some concepts regarding uniform spaces. See [Is64, Wi70] for details.

A *cover* of a set X is a family of subsets $\mathcal{U} \subseteq \mathcal{P}(X)$ such that $\bigcup \mathcal{U} = X$. Let $\mathcal{U}$ and $\mathcal{V}$ be covers of X. We say that $\mathcal{U}$ *refines* $\mathcal{V}$, written $\mathcal{U} \preceq \mathcal{V}$, if for every $U \in \mathcal{U}$ there exists $V \in \mathcal{V}$ with $U \subseteq V$. The *meet* of $\mathcal{U}$ and $\mathcal{V}$ is given by $\mathcal{U} \wedge \mathcal{V} := \{U \cap V : U \in \mathcal{U} \text{ and } V \in \mathcal{V}\}$. For $x, y \in X$, we write $|x-y| < \mathcal{U}$ if $x, y \in U$ for some $U \in \mathcal{U}$. Given any $A \subseteq X$, we define $\mathsf{star}(A; \mathcal{U}) := \{q \in X : |p-q| < \mathcal{U} \text{ for some } p \in A\}$. The *star closure* of $\mathcal{U}$ defined by $\mathcal{U}^* := \{\mathsf{star}(U; \mathcal{U}) : U \in \mathcal{U}\}$ is also a cover of X.

Definition 1. *A family μ of covers of X is called a* uniformity *if it satisfies the following:*

(U1) *if $\mathcal{U}, \mathcal{V} \in \mu$, then $\mathcal{U} \wedge \mathcal{V} \in \mu$;*
(U2) *if $\mathcal{U} \in \mu$ and $\mathcal{U} \preceq \mathcal{V}$, then $\mathcal{V} \in \mu$;*
(U3) *if $\mathcal{U} \in \mu$, then there is a $\mathcal{V} \in \mu$ with $\mathcal{V}^* \preceq \mathcal{U}$ ($\mathcal{V}$* star-refines *$\mathcal{U}$).*

A uniformity μ is said to be Hausdorff *if it additionally satisfies the condition*

(U4) *if $|x - y| < \mathcal{U}$ for every $\mathcal{U} \in \mu_X$, then $x = y$.*

A set equipped with a (Hausdorff) uniformity is called a (Hausdorff) uniform space.

The reader may be more familiar with an equivalent definition based on *entourages*, which we here call the *diagonal uniformity*. Given any diagonal uniformity $\Phi \subseteq \mathcal{P}(X \times X)$, one can construct a uniformity in our definition generated by the family of all the covers $\{A[x] : x \in X\}$, where $A \in \Phi$ and $A[x] := \{y \in X : (x, y) \in A\}$. Conversely, given any uniformity μ in our sense, one can construct a diagonal uniformity generated by the family of all the entourages of the form $\bigcup\{U \times U : U \in \mathcal{U}\}$ with $\mathcal{U} \in \mu$. In this paper, we adopt the covering definition to emphasize the correspondence between *co-total anti-cliques* and *uniform covers* (see Sect. 3).

Let $\mathbb{X} = (X, \mu_X)$ and $\mathbb{Y} = (Y, \mu_Y)$ be uniform spaces. A *uniformly continuous* function from $\mathbb{X}$ to $\mathbb{Y}$ is a function $f : X \to Y$ satisfying that for any $\mathcal{V} \in \mu_Y$ there exists $\mathcal{U} \in \mu_X$ such that $|x - y| < \mathcal{U} \implies |f(x) - f(y)| < \mathcal{V}$ for every $x, y \in X$.

A *(uniform) basis* of μ is a subfamily $\beta \subseteq \mu$ such that for any $\mathcal{U} \in \mu$ there exists $\mathcal{V} \in \beta$ with $\mathcal{V} \preceq \mathcal{U}$. A *(uniform) sub-basis* of μ is a subfamily $\sigma \subseteq \mu$ such that the finite meets of covers in σ form a basis: for any $\mathcal{U} \in \mu$ there exist finitely many $\mathcal{V}_1, \ldots, \mathcal{V}_n \in \sigma$ with $\mathcal{V}_1 \wedge \cdots \wedge \mathcal{V}_n \preceq \mathcal{U}$. Notice that if a family of covers satisfies the conditions **(U2)** and **(U3)** (resp. **(U3)**), it uniquely generates a uniformity as a basis (resp. sub-basis). Each metric space $\mathbb{X} = (X, d)$ has a canonical Hausdorff uniformity, generated by a countable basis $\mathcal{U}_n := \{\mathcal{B}(x; 2^{-n}) : x \in \mathbb{X}\}(n = 1, 2, \ldots)$, where $\mathcal{B}(x; 2^{-n})$ is the open ball of center x and radius 2^{-n}. It is known that a Hausdorff uniformity is induced by some metric if and only if it has a countable basis.

Each uniform space $\mathbb{X} = (X, \mu_X)$ can be equipped with a topological structure, called the *uniform topology*, for which the neighborhood filter at $p \in X$ is given by $\{\mathsf{star}(\{p\}; \mathcal{U}) : \mathcal{U} \in \mu_X\}$. Equivalently, a set $O \subseteq X$ is open with respect to the uniform topology iff for any $p \in O$ there exists $\mathcal{U} \in \mu_X$ such that $\mathsf{star}(\{p\}; \mathcal{U}) \subseteq O$. We will denote by $\tau_{\mathsf{ut}}(\mu)$ the uniform topology induced by μ, which is known to be *completely regular*. Uniform continuity trivially implies topological continuity w.r.t. the uniform topologies. We say that a uniformity μ is *compatible* with a topology τ if $\tau = \tau_{\mathsf{ut}}(\mu)$, and a topological space is *uniformizable* if there exists a uniformity which is compatible with the topology. It is known that uniformizable (resp. Hausdorff uniformizable) topological spaces are exactly completely regular (resp. Tychonoff) spaces.

Each completely regular space $\mathbb{X} = (X, \tau)$ can be equipped with the *finest* uniformity μ_{fine}, the uniformity which contains all of the uniformities compatible with τ. A *fine* uniform space is a uniform space whose uniformity is finest w.r.t. its uniform topology. We denote by $\mathbb{X}_{\text{fine}} = (X, \mu_{\text{fine}})$ the fine uniform space induced by a completely regular space $\mathbb{X}$.

The finest uniformity is characterized by the following property. We denote by **CReg** the category of completely regular spaces and continuous functions, and by **Unif** the category of uniform spaces and uniformly continuous functions. The *fine* functor $F : \mathbf{CReg} \longrightarrow \mathbf{Unif}$, which assigns to each completely regular space $\mathbb{X}$ the finest uniformity, is left adjoint to the *topologizing* functor G :

Unif $\longrightarrow$ **CReg**, which assigns to each uniform space $\mathbb{Y}$ the uniform topology:

$$\mathbf{CReg} \underset{G}{\overset{F}{\rightleftarrows}} \mathbf{Unif} \quad (\perp) \,. \tag{1}$$

Thus, for every completely regular space $\mathbb{X}$ and uniform space $\mathbb{Y}$, $f : \mathbb{X} \to \mathbb{Y}_{\mathsf{ut}}$ is continuous iff $f : \mathbb{X}_{\mathsf{fine}} \to \mathbb{Y}$ is uniformly continuous.

2.2 Coherence Spaces

We here recall some basics of coherence spaces. See [Gi87, Me09] for further information.

Definition 2. *A* coherence space $\boldsymbol{X} = (X, \frown)$ *is a reflexive undirected graph: a set* X *of* tokens *endowed with a reflexive symmetric binary relation* $\frown$ *on* X*, called* coherence.

Throughout this paper, we assume that the token set X is countable. This assumption is quite reasonable in practice to think of tokens as computational objects (see [As90] for the study on computability over coherence spaces).

Several variants of coherence are defined. Given tokens $x, y \in X$, we write $x \smile y$ (*strict coherence*) if $x \frown y$ and $x \neq y$. Notice that coherence and strict coherence are mutually definable from each other.

A *clique* of a coherence space $\boldsymbol{X}$ is a set $a \subseteq X$ such that $x \frown y$ for every $x, y \in a$. An *anti-clique* of $\boldsymbol{X}$ is a set $\mathfrak{c} \subseteq X$ such that $\neg(x \smile y)$ for every $x, y \in \mathfrak{c}$. By abuse of notation, we identify $\boldsymbol{X}$ with the set of all cliques of $\boldsymbol{X}$. We also use $\boldsymbol{X}_{\mathsf{fin}}$, $\boldsymbol{X}_{\mathsf{max}}$ and $\boldsymbol{X}^{\perp}$ for the sets of all finite, maximal and anti-cliques of $\boldsymbol{X}$, respectively. The coherence relation $\frown$ on X can be extended to $\boldsymbol{X}$ by $a \frown b \iff a \cup b \in \boldsymbol{X}$ for $a, b \in \boldsymbol{X}$.

The set of cliques $\boldsymbol{X}$ ordered by inclusion forms a Scott domain whose compact elements are finite cliques of $\boldsymbol{X}$. The *Scott topology* τ_{Sco} on $\boldsymbol{X}$ is generated by $\{\langle a \rangle : a \in \boldsymbol{X}_{\mathsf{fin}}\}$ as a basis, where $\langle a \rangle := \{b \in \boldsymbol{X} : a \subseteq b\}$. Given any set of cliques $\mathcal{A} \subseteq \boldsymbol{X}$, we also write τ_{Sco} for the induced subspace topology on $\mathcal{A}$. Note that $\boldsymbol{X}$ is a T_0-space, and is countably-based due to the assumption that the token set X is countable.

Coherence spaces have a sufficiently rich structure to *represent* abstract spaces. Let us first begin with a coherence space for the real line $\mathbb{R}$.

Example 2 (coherence space for real numbers). Let $\mathbb{D} := \mathbb{Z} \times \mathbb{N}$, where each pair $(m, n) \in \mathbb{D}$ is identified with the *dyadic rational number* $m/2^n$. We use the following notations. For $x = (m, n) \in \mathbb{D}$, we define $\mathsf{den}(x) := n$ (the exponent of the denominator) and $[x] := [(m-1)/2^n; (m+1)/2^n]$ (the compact interval with center x and width $2^{-(n-1)}$), and denote by $\mathbb{D}_n$ the set $\{x \in \mathbb{D} \mid \mathsf{den}(x) = n\}$ for each $n \in \mathbb{N}$.

Let $\mathbf{R}$ be a coherence space $(\mathbb{D}, \frown)$ defined by $x \frown y$ iff $\mathsf{den}(x) \neq \mathsf{den}(y)$ and $[x] \cap [y] \neq \varnothing$. The latter condition immediately implies the inequality $|x - y| \leq 2^{-\mathsf{den}(x)} + 2^{-\mathsf{den}(y)}$, hence each maximal clique $a \in \mathbf{R}_{\mathsf{max}}$ corresponds to a *rapidly-converging* Cauchy sequence $\{x_n\}_{n\in\mathbb{N}}$ (i.e., a sequence of dyadic rationals such that $x_n \in \mathbb{D}_n$ and $|x_n - x_m| \leq 2^{-n} + 2^{-m}$ for every $n, m \in \mathbb{N}$).

A *representation* of the real line $\mathbb{R}$ is a partial surjective map $\phi :\subseteq \boldsymbol{X} \to \mathbb{R}$ from some coherence space $\boldsymbol{X}$. We then have a representation $\rho_{\boldsymbol{R}}$ of $\mathbb{R}$ defined by $\rho_{\boldsymbol{R}}(a) := \lim_{n\to\infty} x_n$ for all $a \in \boldsymbol{R}_{\mathsf{max}}$.

Definition 3 (stable and linear maps). *Let $\boldsymbol{X}$ and $\boldsymbol{Y}$ be coherence spaces. A function $F : \boldsymbol{X} \to \boldsymbol{Y}$ between the set of cliques is said to be* stable*, written $F : \boldsymbol{X} \longrightarrow_{st} \boldsymbol{Y}$, if it is Scott-continuous and $a \frown b \Longrightarrow F(a \cap b) = F(a) \cap F(b)$ for any $a, b \in \boldsymbol{X}$.*

A function $F : \boldsymbol{X} \to \boldsymbol{Y}$ is said to be linear*, written $F : \boldsymbol{X} \longrightarrow_{lin} \boldsymbol{Y}$, if it satisfies that $a = \sum_{i\in I} a_i \Longrightarrow F(a) = \sum_{i\in I} F(a_i)$ for any clique $a \in \boldsymbol{X}$ and any family of cliques $\{a_i\}_{i\in I} \subseteq \boldsymbol{X}$. Here $\sum$ means the disjoint union of cliques.*

It is easy to check that linearity implies stability.

There are alternative definitions of stable and linear maps. Given any function $F : \boldsymbol{X} \longrightarrow \boldsymbol{Y}$ between the set of cliques, we call $(a, y) \in \boldsymbol{X}_{\mathsf{fin}} \times Y$ a *minimal pair* of F if $F(a) \ni y$ and there is no proper subset $a' \subsetneq a$ such that $F(a') \ni y$. We denote by $\mathsf{tr}(F)$ the set of all minimal pairs, called the *trace* of F. Now, F is stable iff it is $\subseteq$-monotone and satisfies that: if $F(a) \ni y$, there is a *unique finite clique* $a_0 \subseteq a$ such that $(a_0, y) \in \mathsf{tr}(F)$.

If F is furthermore linear, the finite clique $a_0 \in \boldsymbol{X}_{\mathsf{fin}}$ must be a singleton. Thus F is linear iff it is $\subseteq$-monotone and satisfies that: if $F(a) \ni y$, there is a *unique token* $x \in a$ such that $(\{x\}, y) \in \mathsf{tr}(F)$. By abuse of notation, we simply write $\mathsf{tr}(F)$ for the set $\{(x, y) | (\{x\}, y) \in \mathsf{tr}(F)\}$ if F is supposed to be linear.

Below are some typical constructions of coherence spaces. Let $\boldsymbol{X}_i = (X_i, \frown_i)$ be a coherence space for $i = 1, 2$. We define:

- $\mathbf{1} = \bot := (\{\bullet\}, \{(\bullet, \bullet)\})$.
- $\boldsymbol{X}_1 \otimes \boldsymbol{X}_2 := (X_1 \times X_2, \frown)$, where $(z, x) \frown (w, y)$ holds iff both $z \frown_1 w$ and $x \frown_2 y$.
- $\boldsymbol{X}_1 \multimap \boldsymbol{X}_2 := (X_1 \times X_2, \frown)$, where $(z, x) \frown (w, y)$ holds iff $z \frown_1 w$ implies $x \frown_2 y$.
- $! \boldsymbol{X}_1 := ((\boldsymbol{X}_1)_{\mathsf{fin}}, \frown)$, where $a \frown b$ holds iff $a \frown_1 b$.

We omit the definitions of additives ($\&$ and $\top$).

In some literature, the exponential coherence space $! \boldsymbol{X}$ is defined on the set of finite *multi-cliques* of $\boldsymbol{X}$. However, we prefer set-cliques, since we will later restrict cliques to be $\subseteq$-minimal (*strict*) ones and such cliques are eventually ordinary sets.

A notable feature of coherence spaces is that they have two closed structures: the *cartesian closed* category **Stab** of coherence spaces and stable maps, and the $*$*-autonomous* category **Lin** of coherence spaces and linear maps equipped with $(\mathbf{1}, \otimes, \multimap, \bot)$. Moreover, one can see $!$ as a linear exponential comonad on the

category **Lin** whose co-Kleisli category is isomorphic to **Stab**. In fact, a stable map $F : \boldsymbol{X} \longrightarrow_{st} \boldsymbol{Y}$ can be identified with a linear map $G : !\,\boldsymbol{X} \longrightarrow_{lin} \boldsymbol{Y}$ so that $\mathsf{tr}(F) = \mathsf{tr}(G) \subseteq \boldsymbol{X}_{\mathsf{fin}} \times Y$. This leads to a linear-non-linear adjunction:

$$\mathbf{Stab} \underset{L}{\overset{K}{\underset{\longleftarrow}{\overset{\longrightarrow}{\bot}}}} \mathbf{Lin}\,. \tag{2}$$

The purpose of this paper is to establish a connection between the two adjunctions (1) and (2), which will be done in Sect. 3.2 by giving a *pseudo-map of adjunctions* between them.

We do not describe anymore the categorical structures of coherence spaces, but let us just mention the following. Given any linear map $F : \boldsymbol{X} \longrightarrow_{lin} \boldsymbol{Y}$, its trace forms the clique $\mathsf{tr}(F) \in \boldsymbol{X} \multimap \boldsymbol{Y}$. Conversely, given any clique $\kappa \in \boldsymbol{X} \multimap \boldsymbol{Y}$, the linear map $\widehat{\kappa} : \boldsymbol{X} \longrightarrow_{lin} \boldsymbol{Y}$ is induced as $\widehat{\kappa}(a) := \{y \in Y : (x, y) \in \kappa \text{ for some } x \in a\}$.

3 Uniform Structures on Coherence Spaces with Totality

In this section, we introduce the notions of (co-)totality of (anti-)cliques. We then observe that uniform structures are induced on the set of strict (i.e. $\subseteq$-minimal) total cliques by the corresponding strict co-totalities.

3.1 Coherence Spaces with Totality

Let $\boldsymbol{X}$ be a coherence space. Recall that given any clique $a \in \boldsymbol{X}$ and any anti-clique $\mathfrak{c} \in \boldsymbol{X}^{\bot}$, $a \cap \mathfrak{c}$ is either empty or a singleton. If the latter is the case, we write $a \perp \mathfrak{c}$.

Given any set of cliques $\mathcal{A} \subseteq \boldsymbol{X}$, we denote by $\mathcal{A}^{\bot}$ the set of anti-cliques $\{\mathfrak{c} \in \boldsymbol{X}^{\bot} : \forall a \in \mathcal{A}.\ a \perp \mathfrak{c}\}$, called the *orthogonal* of $\mathcal{A}$. One can immediately observe the following: (i) $\mathcal{A} \subseteq \mathcal{A}^{\bot\bot} \subseteq \boldsymbol{X}$; (ii) $\mathcal{B} \subseteq \mathcal{A}$ implies $\mathcal{A}^{\bot} \subseteq \mathcal{B}^{\bot}$; and (iii) $\mathcal{A}^{\bot} = \mathcal{A}^{\bot\bot\bot}$. As a consequence, $\mathcal{A} = \mathcal{A}^{\bot\bot}$ iff it is the orthogonal of some $\mathcal{B} \subseteq \boldsymbol{X}^{\bot}$.

Definition 4 (coherence spaces with totality). *A* coherence space with totality *is a coherence space $\boldsymbol{X}$ equipped with a set of cliques $\mathcal{T}_{\boldsymbol{X}} \subseteq \boldsymbol{X}$ such that $\mathcal{T}_{\boldsymbol{X}} = \mathcal{T}_{\boldsymbol{X}}^{\bot\bot}$, called a* totality. *Cliques in $\mathcal{T}_{\boldsymbol{X}}$ are said to be* total.

The orthogonal $\mathcal{T}_{\boldsymbol{X}}^{\bot}$ is called the co-totality, *and its members are called* co-total anti-cliques.

It is clear that a totality $\mathcal{T}_{\boldsymbol{X}}$ is upward-closed with respect to $\subseteq$, and is closed under compatible intersections: if $a, b \in \mathcal{T}_{\boldsymbol{X}}$ and $a \frown b$ then $a \cap b \in \mathcal{T}_{\boldsymbol{X}}$. As a consequence, every total clique $a \in \mathcal{T}_{\boldsymbol{X}}$ is associated with a unique minimal part $a^{\circ} := \bigcap\{b \in \mathcal{T}_{\boldsymbol{X}} : b \subseteq a\}$ which is again total. Such a total clique is called

strict (or *material* in the sense of ludics). We write $\mathcal{T}_{\boldsymbol{X}}^{\circ}$ for the set of strict total cliques of $\boldsymbol{X}$. We have $\mathcal{T}_{\boldsymbol{X}} = \{b \in \boldsymbol{X} : a \subseteq b \text{ for some } a \in \mathcal{T}_{\boldsymbol{X}}^{\circ}\} = (\mathcal{T}_{\boldsymbol{X}}^{\circ})^{\perp\perp}$, thus defining a totality is essentially equivalent to defining a strict totality. A more direct definition of $\mathcal{T}_{\boldsymbol{X}}^{\circ}$ is as follows: $a \in \mathcal{T}_{\boldsymbol{X}}^{\circ}$ iff for every $\mathfrak{c} \in (\mathcal{T}_{\boldsymbol{X}}^{\perp})^{\circ}$ there exists $x \in a$ such that $x \in \mathfrak{c}$ (totality), and dually, for every $x \in a$ there exists $\mathfrak{c} \in (\mathcal{T}_{\boldsymbol{X}}^{\perp})^{\circ}$ such that $x \in \mathfrak{c}$ (strictness).

Our use of totality is directly inspired by Kristiansen and Normann [KN97], although their use does not take into account strictness and bi-orthogonality. Similar constructions are abundant in the literature, eg., *totality spaces* by Loader [Loa94] and *finiteness spaces* by Ehrhard [Ehr05].

Example 3. Consider the coherence space $\boldsymbol{R} = (\mathbb{D}, \frown)$ for real numbers defined in Example 2. The set of maximal cliques $\mathcal{T}_{\boldsymbol{R}} := \boldsymbol{R}_{\mathsf{max}}$ is a totality on $\boldsymbol{R}$: it is easy to see that $\mathcal{T}_{\boldsymbol{R}}^{\perp} = \{\mathbb{D}_n : n \in \mathbb{N}\}$, hence $\mathcal{T}_{\boldsymbol{R}}^{\perp\perp} = \boldsymbol{R}_{\mathsf{max}} = \mathcal{T}_{\boldsymbol{R}}$. We also have $\mathcal{T}_{\boldsymbol{R}}^{\circ} = \boldsymbol{R}_{\mathsf{max}}$ due to maximality.

Example 4. The idea of Example 2 can be generalized to a more general class. Let $\mathbb{X} = (X, \mu)$ be a Hausdorff uniform space with a countable basis $\beta = \{\mathcal{U}_n\}_{n\in\mathbb{N}}$ consisting of countable covers. A form of the *metrization theorem* states that such a space must be separable metrizable (see [Ke75] for instance).

Let B be the set $\{(n, U) : n \in \mathbb{N}, U \in \mathcal{U}_n\}$. We define a coherence space $\boldsymbol{B}_{\mathbb{X},\beta} = (B, \frown)$ by $(n, U) \frown (m, V)$ iff $n \neq m$ and $U \cap V \neq \varnothing$ for all $(n, U), (m, V) \in B$. Each $a \in (\boldsymbol{B}_{\mathbb{X},\beta})_{\mathsf{max}}$ corresponds to a sequence of sets $\{U_n\}_{n\in\mathbb{N}}$ such that $U_n \in \mathcal{U}_n$ and $U_n \cap U_m \neq \varnothing$ for every $n, m \in \mathbb{N}$. By the Hausdorff property, it indicates *at most* one point in $\mathbb{X}$. It is easy to see that $\mathcal{T}_{\boldsymbol{B}_{\mathbb{X},\beta}} := (\boldsymbol{B}_{\mathbb{X},\beta})_{\mathsf{max}}$ is a totality on $\boldsymbol{B}_{\mathbb{X},\beta}$.

The separable metrizable space $\mathbb{X}$ has a "standard" representation $\delta_{\mathbb{X}} :\subseteq \boldsymbol{B}_{\mathbb{X},\beta} \to \mathbb{X}$ defined by $\delta_{\mathbb{X}}(a) := p$ if $p \in \bigcap_{n\in\mathbb{N}} U_n \neq \varnothing$ and undefined otherwise for every $a = \{U_n\}_{n\in\mathbb{N}} \in (\boldsymbol{B}_{\mathbb{X},\beta})_{\mathsf{max}}$. The definition of $\delta_{\mathbb{X}}$ does not depend on the choice of the countable basis β, up to isomorphisms in the category of representations (see Sect. 4), so we will below drop the index β from the notation.

All constructions of coherence spaces are extended to totalities in a rather canonical way. Let $(\boldsymbol{X}, \mathcal{T}_{\boldsymbol{X}})$ and $(\boldsymbol{Y}, \mathcal{T}_{\boldsymbol{Y}})$ be coherence spaces with totality. Define:

- $\mathcal{T}_{\boldsymbol{X}^{\perp}} := \mathcal{T}_{\boldsymbol{X}}^{\perp}$; $\mathcal{T}_{\mathbf{1}} := \mathbf{1}_{\mathsf{max}}$.
- $\mathcal{T}_{\boldsymbol{X}\otimes\boldsymbol{Y}} := (\mathcal{T}_{\boldsymbol{X}} \otimes \mathcal{T}_{\boldsymbol{Y}})^{\perp\perp}$, where $\mathcal{T}_{\boldsymbol{X}} \otimes \mathcal{T}_{\boldsymbol{Y}}$ consists of $a \otimes b := \{(x, y) : x \in a, y \in b\}$ for all $a \in \mathcal{T}_{\boldsymbol{X}}$ and $b \in \mathcal{T}_{\boldsymbol{Y}}$.
- $\mathcal{T}_{\boldsymbol{X}\multimap\boldsymbol{Y}} := \{\kappa \in (\boldsymbol{X}_1 \multimap \boldsymbol{X}_2) : \widehat{\kappa}[\mathcal{T}_{\boldsymbol{X}}] \subseteq \mathcal{T}_{\boldsymbol{Y}}\}$.
- $\mathcal{T}_{!\boldsymbol{X}} := (!\mathcal{T}_{\boldsymbol{X}})^{\perp\perp}$, where $!\mathcal{T}_{\boldsymbol{X}}$ consists of $!a := \{a_0 \in \boldsymbol{X} : a_0 \subseteq_{\mathsf{fin}} a\}$ for all $a \in \mathcal{T}_{\boldsymbol{X}}$.

The connectives $\otimes$ and $!$ admit "internal completeness" in the following sense.

Proposition 1. *For any coherence spaces with totality,* $(\mathcal{T}_{\boldsymbol{X}} \otimes \mathcal{T}_{\boldsymbol{Y}})^{\perp\perp\circ} = \mathcal{T}_{\boldsymbol{X}}^{\circ} \otimes \mathcal{T}_{\boldsymbol{Y}}^{\circ}$ *and* $(!\mathcal{T}_{\boldsymbol{Z}})^{\perp\perp\circ} = !(\mathcal{T}_{\boldsymbol{Z}}^{\circ})$ *hold.*

A proof is given in the full version. This property, which is essential for the main result of this section, seems very specific to our choice of coherence spaces model.

Let us give the definition of *total* functions.

Definition 5. *A linear map $F : \boldsymbol{X} \longrightarrow_{lin} \boldsymbol{Y}$ is called* total *if* $\mathsf{tr}(F) \in \mathcal{T}_{\boldsymbol{X} \multimap \boldsymbol{Y}}$*, or equivalently if F preserves totality: $F[\mathcal{T}_{\boldsymbol{X}}] \subseteq \mathcal{T}_{\boldsymbol{Y}}$.*

Similarly, a stable map $F : \boldsymbol{X} \longrightarrow_{st} \boldsymbol{Y}$ is called total *if it preserves totality.*

Theorem 6. *The category $\mathbf{Lin}_{\mathsf{Tot}}$ of coherence spaces with totality and total linear maps is a model of classical linear logic (i.e., a $*$-autonomous category with finite (co)products and a linear exponential (co)monad).*

The proof is essentially due to [HS03].

The category $\mathbf{Stab}_{\mathsf{Tot}}$ of coherence spaces with totality and total stable maps is trivially the co-Kleisli category of the linear exponential comonad ! and hence is cartesian closed. As a consequence, we again have a linear-non-linear adjunction between $\mathbf{Stab}_{\mathsf{Tot}}$ and $\mathbf{Lin}_{\mathsf{Tot}}$.

3.2 Uniformities Induced by Co-Totality

We shall next show that totalities on coherence spaces can be equipped with uniform structures *induced by co-totalities*. Our claim can be summarized as follows. Given any coherence space with totality $(\boldsymbol{X}, \mathcal{T}_{\boldsymbol{X}})$, its strict totality $\mathcal{T}^{\circ}_{\boldsymbol{X}}$ is endowed with the Scott topology τ_{Sco}; it is simultaneously endowed with the Hausdorff uniformities induced by the strict co-totalities $(\mathcal{T}^{\perp}_{\boldsymbol{X}})^{\circ}$ and $(\mathcal{T}^{\perp}_{!\boldsymbol{X}})^{\circ}$ which are compatible with τ_{Sco}; and moreover the latter uniformity is the *finest*.

Recall that each finite clique $a \in \boldsymbol{X}_{\mathsf{fin}}$ generates the upper set $\langle a \rangle := \{b \in \boldsymbol{X} : b \supseteq a\}$ in such a way that "incoherence" corresponds to disjointness:

$$\neg(x \frown y) \iff \langle x \rangle \cap \langle y \rangle = \varnothing; \qquad \neg(a \frown b) \iff \langle a \rangle \cap \langle b \rangle = \varnothing$$

for every $x, y \in X$ and $a, b \in \boldsymbol{X}$, where $\langle x \rangle$ stands for $\langle \{x\} \rangle$ by abuse of notation. Let us use the notations $\langle x \rangle^{\circ} := \langle x \rangle \cap \mathcal{T}^{\circ}_{\boldsymbol{X}}$ and $\langle a \rangle^{\circ} := \langle a \rangle \cap \mathcal{T}^{\circ}_{\boldsymbol{X}}$.

A strict co-total anti-clique of $\boldsymbol{X}$ is called a *uni-cover* of $\mathcal{T}^{\circ}_{\boldsymbol{X}}$. This terminology can be explained as follows. As noted above, an anti-clique $\mathfrak{c} \in \boldsymbol{X}^{\perp}$ can be seen as a disjoint family of upper sets: $\{\langle x \rangle^{\circ} : x \in \mathfrak{c}\}$. Co-totality of $\mathfrak{c}$ then means that every $a \in \mathcal{T}^{\circ}_{\boldsymbol{X}}$ is contained in some $\langle x \rangle^{\circ}$ with $x \in \mathfrak{c}$. Thus $\mathcal{T}^{\circ}_{\boldsymbol{X}} = \sum_{x \in \mathfrak{c}} \langle x \rangle^{\circ}$. Moreover, $\mathfrak{c}$ being strict means that $\langle x \rangle^{\circ}$ is nonempty for every $x \in \mathfrak{c}$. That is, restricting $\mathfrak{c} \in \mathcal{T}^{\perp}_{\boldsymbol{X}}$ to $\mathfrak{c}^{\circ} \in (\mathcal{T}^{\perp}_{\boldsymbol{X}})^{\circ}$ amounts to removing all empty $\langle x \rangle^{\circ}$ from the disjoint cover $\{\langle x \rangle^{\circ} : x \in \mathfrak{c}\}$.

On the other hand, each $\mathfrak{C} \in (\mathcal{T}^{\perp}_{!\boldsymbol{X}})^{\circ}$ is called an *unbounded-cover* of $\mathcal{T}^{\circ}_{\boldsymbol{X}}$. It is also identified with a disjoint cover $\{\langle a \rangle^{\circ} : a \in \mathfrak{C}\}$ of $\mathcal{T}^{\circ}_{\boldsymbol{X}}$, consisting of nonempty upper sets, so that $\mathcal{T}^{\circ}_{\boldsymbol{X}} = \sum_{a \in \mathfrak{C}} \langle a \rangle^{\circ}$.

To emphasize the uniformity aspect, we will use the notations $\sigma^{\mathrm{b}}_{\boldsymbol{X}} := (\mathcal{T}^{\perp}_{\boldsymbol{X}})^{\circ}$ and $\beta^{\mathrm{ub}}_{\boldsymbol{X}} := (\mathcal{T}^{\perp}_{!\boldsymbol{X}})^{\circ}$. Each uni-cover can be considered as an unbounded-cover consisting of singletons: $\sigma^{\mathrm{b}}_{\boldsymbol{X}} \subseteq \beta^{\mathrm{ub}}_{\boldsymbol{X}}$ by $\mathfrak{c} \in \sigma^{\mathrm{b}}_{\boldsymbol{X}} \mapsto \{\{x\} : x \in \mathfrak{c}\} \in \beta^{\mathrm{ub}}_{\boldsymbol{X}}$.

The families $\sigma^{\mathrm{b}}_{\boldsymbol{X}}$ and $\beta^{\mathrm{ub}}_{\boldsymbol{X}}$ indeed generate uniformities on $\mathcal{T}^{\circ}_{\boldsymbol{X}}$:

Proposition 2. *$(\mathcal{T}^{\circ}_{\boldsymbol{X}}, \beta^{\mathrm{ub}}_{\boldsymbol{X}})$ satisfies axioms* **(U1)**, **(U3)** *and* **(U4)**, *while $(\mathcal{T}^{\circ}_{\boldsymbol{X}}, \sigma^{\mathrm{b}}_{\boldsymbol{X}})$ satisfies* **(U3)** *and* **(U4)** *in Definition 1.*

Proof. **(U1)** First notice that $\beta^{\mathrm{ub}}_{\boldsymbol{X}} = (\mathcal{T}^{\perp}_{!\boldsymbol{X}})^{\circ} = (!\,\mathcal{T}_{\boldsymbol{X}})^{\perp\perp\perp\circ} = (!\,\mathcal{T}_{\boldsymbol{X}})^{\perp\circ}$. Given $\mathfrak{A}, \mathfrak{B} \in (!\,\mathcal{T}_{\boldsymbol{X}})^{\perp}$, let $\mathfrak{A} \wedge \mathfrak{B} := \{a \cup b \ : \ a \in \mathfrak{A}, b \in \mathfrak{B} \mathrm{and} a \supset b\}$, which is indeed the meet of the covers $\mathfrak{A}$ and $\mathfrak{B}$ under the identification of co-total anti-cliques with disjoint covers, since $\langle a \cup b\rangle = \langle a\rangle \cap \langle b\rangle$ holds for all $a, b \in \boldsymbol{X}$.

All we have to check is that $\mathfrak{A} \wedge \mathfrak{B}$ belongs to $(!\,\mathcal{T}_{\boldsymbol{X}})^{\perp}$. Then it easily follows that if $\mathfrak{A}, \mathfrak{B}$ belong to $\beta^{\mathrm{ub}}_{\boldsymbol{X}}$, so does $(\mathfrak{A} \wedge \mathfrak{B})^{\circ}$. Given $c \in \mathcal{T}_{\boldsymbol{X}}$ so that $!c \in !\,\mathcal{T}_{\boldsymbol{X}}$, there are $a \in \mathfrak{A}$ and $b \in \mathfrak{B}$ such that $a \in !c$ and $b \in !c$. Hence $a \cup b \in !c \cap (\mathfrak{A} \wedge \mathfrak{B})$.

(U3) In general, we have $\mathsf{star}(U; \mathfrak{C}) = U$ for any disjoint cover $\mathfrak{C}$ of $\mathcal{T}^{\circ}_{\boldsymbol{X}}$ and $U \in \mathfrak{C}$. Hence each $\mathfrak{A} \in \beta^{\mathrm{ub}}_{\boldsymbol{X}}$, which is disjoint, star-refines itself.

(U4) Assume that $a, b \in \mathcal{T}^{\circ}_{\boldsymbol{X}}$ with $a \neq b$. Then there are $x \in a \backslash b$ and $\mathfrak{c} \in \sigma^{\mathrm{b}}_{\boldsymbol{X}} = (\mathcal{T}^{\perp}_{\boldsymbol{X}})^{\circ}$ such that $x \in \mathfrak{c}$ by strictness of a. As $a \in \langle x\rangle^{\circ}$, $b \notin \langle x\rangle^{\circ}$ and $\mathfrak{c}$ is a disjoint cover, this witnesses the Hausdorff property for $\sigma^{\mathrm{b}}_{\boldsymbol{X}}$ (so for $\beta^{\mathrm{ub}}_{\boldsymbol{X}}$ too). ■

Consequently, $\beta^{\mathrm{ub}}_{\boldsymbol{X}}$, as uniform basis, generates a Hausdorff uniformity $\mu^{\mathrm{ub}}_{\boldsymbol{X}}$ called the *unbounded uniformity*, while $\sigma^{\mathrm{b}}_{\boldsymbol{X}}$, as uniform sub-basis, generates another Hausdorff uniformity $\mu^{\mathrm{b}}_{\boldsymbol{X}} \subseteq \mu^{\mathrm{ub}}_{\boldsymbol{X}}$ called the *bounded uniformity*. The index $\boldsymbol{X}$ will be often dropped if it is obvious from the context.

Unlike $\beta^{\mathrm{ub}}_{\boldsymbol{X}}$, the family $\sigma^{\mathrm{b}}_{\boldsymbol{X}}$ is not closed under finite meets. To make it closed, we have to extend it to $\beta^{\mathrm{b}}_{\boldsymbol{X}} \subseteq \beta^{\mathrm{ub}}_{\boldsymbol{X}}$, which consists of all finite meets of uni-covers: $\mathfrak{c}_1 \wedge \cdots \wedge \mathfrak{c}_m := \{\{x_1, \ldots, x_m\} \in \boldsymbol{X} : x_i \in \mathfrak{c}_i \ (1 \leq i \leq m)\}^{\circ}$. Notice that $\mathfrak{c}_1 \wedge \cdots \wedge \mathfrak{c}_m$ consists of cliques of size at most m, while unbounded-covers do not. That is why $\mu^{\mathrm{b}}_{\boldsymbol{X}}$ is called bounded.

Although $\mu^{\mathrm{b}}_{\boldsymbol{X}}$ and $\mu^{\mathrm{ub}}_{\boldsymbol{X}}$ are different as uniformities, they do induce the same uniform topology:

Theorem 7. *The (un)bounded uniformity on $\mathcal{T}^{\circ}_{\boldsymbol{X}}$ is compatible with the Scott topology restricted to $\mathcal{T}^{\circ}_{\boldsymbol{X}}$. That is, $\tau_{\mathsf{ut}}(\mu^{\mathrm{b}}) = \tau_{\mathsf{ut}}(\mu^{\mathrm{ub}}) = \tau_{\mathrm{Sco}}$.*

Proof. By definition, $U \subseteq \mathcal{T}^{\circ}_{\boldsymbol{X}}$ is open with respect to $\tau_{\mathsf{ut}}(\mu^{\mathrm{ub}})$ iff for every $a \in U$ there exists $\mathfrak{A} \in \beta^{\mathrm{ub}}_{\boldsymbol{X}}$ such that $\mathsf{star}(\{a\}; \mathfrak{A}) \subseteq U$ (see Sect. 2.1). Due to disjointness of $\mathfrak{A}$, however, $\mathsf{star}(\{a\}; \mathfrak{A})$ just amounts to $\langle a_0\rangle^{\circ}$, where a_0 is the unique clique in $\mathfrak{A}$ such that $a \in \langle a_0\rangle^{\circ}$. Moreover, any finite clique $a_1 \in \boldsymbol{X}_{\mathsf{fin}}$ with $\langle a_1\rangle^{\circ} \neq \varnothing$ is contained in some $\mathfrak{A} \in \beta^{\mathrm{ub}}_{\boldsymbol{X}}$ (see the full version for the proof). All together, U is open iff for every $a \in U$ there exists $a_0 \in \boldsymbol{X}_{\mathsf{fin}}$ such that $a \in \langle a_0\rangle^{\circ}$ iff U is open with respect to τ_{Sco}.

The same reasoning works for $\tau_{\mathsf{ut}}(\mu^{\mathrm{b}})$ too. ■

The unbounded uniformity μ^{ub} is hence compatible with, and finer than the bounded uniformity μ^{b}. We can furthermore show that it is the finest uniformity on $\mathcal{T}^{\circ}_{\boldsymbol{X}}$. The omitted proof is found in the full version.

Theorem 8. $(\mathcal{T}^{\circ}_{\boldsymbol{X}}, \mu^{\mathrm{ub}}_{\boldsymbol{X}})$ *is a fine uniform space.*

Due to internal completeness (Proposition 1), we have a bijection $\mathcal{T}^{\circ}_{\boldsymbol{X}} \simeq \mathcal{T}^{\circ}_{!\boldsymbol{X}}$ defined by $a \in \mathcal{T}^{\circ}_{\boldsymbol{X}} \leftrightarrow !a \in \mathcal{T}^{\circ}_{!\boldsymbol{X}}$. Notice also that $\beta^{\mathrm{ub}}_{\boldsymbol{X}} = (\mathcal{T}^{\perp}_{!\boldsymbol{X}})^{\circ} = \sigma^{\mathrm{b}}_{!\boldsymbol{X}}$ and fine uniformity is preserved under uniform homeomorphisms. These facts together allow us to prove:

Corollary 1. *The above bijection gives a uniform homeomorphism* $(\mathcal{T}^{\circ}_{\boldsymbol{X}}, \mu^{\mathrm{ub}}_{\boldsymbol{X}}) \simeq (\mathcal{T}^{\circ}_{!\boldsymbol{X}}, \mu^{\mathrm{b}}_{!\boldsymbol{X}})$. *As a consequence,* $(\mathcal{T}^{\circ}_{!\boldsymbol{X}}, \mu^{\mathrm{b}}_{!\boldsymbol{X}})$ *is a fine uniform space.*

Let $(\boldsymbol{X}, \mathcal{T}_{\boldsymbol{X}})$ and $(\boldsymbol{Y}, \mathcal{T}_{\boldsymbol{Y}})$ be coherence spaces with totality and $F : \boldsymbol{X} \longrightarrow_{lin} \boldsymbol{Y}$ be a total linear map. First notice that a total linear map can be seen as a total function between the totalities. Moreover, one can naturally restrict it to the total function between the strict totalities. The *strict total function* $F^{\circ} : \mathcal{T}^{\circ}_{\boldsymbol{X}} \to \mathcal{T}^{\circ}_{\boldsymbol{Y}}$ is defined by $F^{\circ}(a) := (F(a))^{\circ}$ for all $a \in \mathcal{T}^{\circ}_{\boldsymbol{X}}$. This is well-defined and strict total functions are compositional due to minimality of strict total cliques. We are now ready to establish uniform continuity of linear maps.

Theorem 9. *For any* $\mathfrak{b} \in \sigma^{\mathrm{b}}_{\boldsymbol{Y}}$ *there exists* $\mathfrak{a} \in \sigma^{\mathrm{b}}_{\boldsymbol{X}}$ *such that* $|a - b| < \mathfrak{a} \Rightarrow |F(a) - F(b)| < \mathfrak{b}$ *for every* $a, b \in \mathcal{T}^{\circ}_{\boldsymbol{X}}$. *Consequently, (i)* $F^{\circ} : \mathcal{T}^{\circ}_{\boldsymbol{X}} \to \mathcal{T}^{\circ}_{\boldsymbol{Y}}$ *is uniformly continuous w.r.t. the bounded uniformities; and (ii)* $F^{\circ} : \mathcal{T}^{\circ}_{\boldsymbol{X}} \to \mathcal{T}^{\circ}_{\boldsymbol{Y}}$ *is topologically continuous w.r.t. the uniform topologies.*

Proof. Since $\mathbf{Lin}_{\mathsf{Tot}}$ is *-autonomous, one can take the transpose $F^{\perp} : \boldsymbol{Y}^{\perp} \longrightarrow_{lin} \boldsymbol{X}^{\perp}$ defined by $x \in F^{\perp}(\{y\}) \Leftrightarrow F(\{x\}) \ni y$ for every $x \in X$ and $y \in Y$. Let $\mathfrak{a} := F^{\perp}(\mathfrak{b}) \in \boldsymbol{X}^{\perp}$. By definition, one can immediately observe that $a, b \in \langle x \rangle^{\circ}$ with $x \in \mathfrak{a}$ implies $F(a), F(b) \in \langle y \rangle^{\circ}$ with $y \in \mathfrak{b}$. ■

We thus obtain a functor $J : \mathbf{Lin}_{\mathsf{Tot}} \to \mathbf{Unif}$ which sends a coherence space with totality $(\boldsymbol{X}, \mathcal{T}_{\boldsymbol{X}})$ to the (indeed Hausdorff) uniform space $(\mathcal{T}^{\circ}_{\boldsymbol{X}}, \mu^{\mathrm{b}})$ and a total linear map to the corresponding uniformly continuous function which is shown in the above theorem. There is also a functor $I : \mathbf{Stab}_{\mathsf{Tot}} \to \mathbf{CReg}$ sending $(\boldsymbol{X}, \mathcal{T}_{\boldsymbol{X}})$ to the completely regular (indeed Tychonoff) space $(\mathcal{T}^{\circ}_{\boldsymbol{X}}, \tau_{\mathrm{Sco}})$ and a total stable map to the corresponding continuous function. We now have the following diagram, in which the two squares commute (up to natural isomorphisms):

$$\begin{array}{ccc} & F & \\ \mathbf{CReg} & \overset{\perp}{\rightleftarrows} & \mathbf{Unif} \\ & G & \\ I \uparrow & & \uparrow J \\ & K & \\ \mathbf{Stab}_{\mathsf{Tot}} & \overset{\perp}{\rightleftarrows} & \mathbf{Lin}_{\mathsf{Tot}} \\ & L & \end{array} \qquad (3)$$

In addition, one can show that the pair of functors $\langle I, J\rangle$ preserves an adjunction: it is a *pseudo-map of adjunctions* in the sense of Jacobs [Ja99].

This combines (1) and (2), as we have planned.

4 Coherent Representations

In this section, we introduce *coherent representations*, a model of abstract computations based on coherence spaces. Our motivation is to examine how resource-sensitive properties like stability or linearity affect on computations over topological structures.

4.1 Representations as a Realizability Model

We represent abstract spaces largely following the mainstreams of *computable analysis*: the type-two theory of effectivity (TTE) [KW85, We00, BHW08] and the theory of domain representations [Bl97, ES99, SHT08]. In both theories, computations over abstract spaces are first tracked by continuous maps over their base spaces (the Baire space $\mathbb{B} = \mathbb{N}^\omega$ in TTE and Scott domains in domain representations), and then the computability notions are introduced. Following a similar idea, we use coherence spaces to represent topological spaces as base spaces, and track computations by stable maps just as in Examples 2 and 4 (for the issue of computability of stable maps, see [As90]; one can impose *effectivity* on coherence spaces just as in *effective Scott domains*).

Let us give a formal definition:

Definition 10. *Let S be an arbitrary set. A tuple $(\boldsymbol{X}, \rho, S)$ is called a* representation *of S if $\boldsymbol{X}$ is a coherence space and $\rho :\subseteq \boldsymbol{X} \to S$ is a partial surjective map. Below, we write $\boldsymbol{X} \xrightarrow{\rho} S$, or simply ρ, for this representation.*

Definition 11 (stable realizability). *Let $\boldsymbol{X} \xrightarrow{\rho_{\boldsymbol{X}}} S$ and $\boldsymbol{Y} \xrightarrow{\rho_{\boldsymbol{Y}}} T$ be representations of sets S and T. A function $f : S \to T$ is* stably realizable *with respect to $\rho_{\boldsymbol{X}}$ and $\rho_{\boldsymbol{Y}}$ if it is* tracked *by some stable map $F : \boldsymbol{X} \longrightarrow_{st} \boldsymbol{Y}$, that is, the stable map F makes the following diagram commute:*

$$\begin{array}{ccc} \boldsymbol{X} & \xrightarrow{F} & \boldsymbol{Y} \\ {\scriptstyle \rho_{\boldsymbol{X}}}\downarrow & & \downarrow{\scriptstyle \rho_{\boldsymbol{Y}}} \\ S & \xrightarrow{f} & T \end{array} \tag{4}$$

We denote by **StabRep** the category of coherent representations and stably realizable functions.

With the help of the theory of *applicative morphisms* [Lon94, Ba00, Ba02], one can compare **StabRep** with other models of representations, especially with the TTE model. A partial surjective map $\phi :\subseteq \mathbb{B} \to S$ onto a set S is said to

be a *TTE-representation* of S. Given any TTE-representations δ and γ of sets S and T, a function $f : S \to T$ is said to be *continuously realizable* if there exists a continuous function $F :\subseteq \mathbb{B} \to \mathbb{B}$ which tracks f as in the diagram (4). We denote by **TTERep** the category of TTE-representations and continuously realizable functions. We then obtain an *applicative retraction* between coherent representations and TTE-representations. Roughly speaking, one can *embed* the type-two theory of effectivity into the theory of coherent representations.

Theorem 12. *The category* **TTERep** *is equivalent to a full coreflexive subcategory of* **StabRep**.

In [MT16], we have defined *spanned* coherent representations, which forms a full coreflexive subcategory of **StabRep** equivalent to **TTERep**. As a consequence of the theorem, various ideas and results in TTE can be imported to spanned representations. In particular, one can import the notion of *admissible representations*: representations of topological spaces which provide the best way to encode the approximation structures. The next theorem immediately follows from [Sc02]:

Theorem 13 ([MT16]). *Let* $\mathbb{X}$ *and* $\mathbb{Y}$ *be topological spaces represented by admissible spanned representations* $\boldsymbol{X} \xrightarrow{\rho_{\boldsymbol{X}}} \mathbb{X}$ *and* $\boldsymbol{Y} \xrightarrow{\rho_{\boldsymbol{Y}}} \mathbb{Y}$. *A function* $f : \mathbb{X} \longrightarrow \mathbb{Y}$ *is stably realizable if and only if it is* sequentially continuous, *that is, it preserves the limit of any convergent sequence:* $x_n \to x \;\Rightarrow\; f(x_n) \to f(x)$.

For instance, the coherent representation $\boldsymbol{R} \xrightarrow{\rho_{\boldsymbol{R}}} \mathbb{R}$ defined in Example 2 is spanned and admissible. Consequently, a function $f : \mathbb{R} \to \mathbb{R}$ is stably realizable w.r.t. $\rho_{\boldsymbol{R}}$ iff it is continuous (sequential continuity coincides with usual continuity for real functions). This correspondence can be generalized to a vast class of topological spaces (quotients of countably-based (*qcb-*)spaces [Sc02, Si03]).

Notice that given any topological space $\mathbb{X}$, its admissible representations are "interchangeable": if $\boldsymbol{X}_0 \xrightarrow{\rho_0} \mathbb{X}$ and $\boldsymbol{X}_1 \xrightarrow{\rho_1} \mathbb{X}$ are admissible, then the identity map $\mathsf{id} : \mathbb{X} \longrightarrow \mathbb{X}$ is realized by stable maps $F : \boldsymbol{X}_0 \longrightarrow_{st} \boldsymbol{X}_1$ and $G : \boldsymbol{X}_1 \longrightarrow_{st} \boldsymbol{X}_0$ which reduce each representation to another.

4.2 Linear Realizability for Separable Metrizable Spaces

On the other hand, we have found in [MT16] a linear variant of the above equivalence between stable realizability and continuity: a real function $f : \mathbb{R} \to \mathbb{R}$ is *linearly realizable* iff it is *uniformly continuous*. We below try to generalize this correspondence to separable metrizable uniform spaces, based on standard representations defined in Example 4. Recall that given any separable metrizable space $\mathbb{X}$, its standard representation $\boldsymbol{B}_{\mathbb{X}} \xrightarrow{\delta_{\mathbb{X}}} \mathbb{X}$ is constructed from a countable basis of $\mathbb{X}$.

Definition 14 (linear realizability). *Let* $\boldsymbol{X} \xrightarrow{\rho_{\boldsymbol{X}}} S$ *and* $\boldsymbol{Y} \xrightarrow{\rho_{\boldsymbol{Y}}} T$ *be representations. A function* $f : S \to T$ *is* linearly realizable *with respect to* $\rho_{\boldsymbol{X}}$ *and* $\rho_{\boldsymbol{Y}}$ *if it is tracked by a linear map* $F : \boldsymbol{X} \longrightarrow_{lin} \boldsymbol{Y}$ *in the sense of the above diagram (4).*

Theorem 15. *The category* **LinRep** *of coherent representations and linearly realizable functions is a model of intuitionistic linear logic (a* linear category *in the sense of Bierman [Bi94]).*

Proof Sketch. Recall that *linear combinatory algebras (LCA)* are a linear variant of partial combinatory algebras (PCA) [AL00, AHS02]. One can construct various realizability models based on LCAs, and especially, the PER category $\mathbf{PER}(\mathbb{A})$ over an LCA $\mathbb{A}$ is known to be a model of ILL as shown in [AL05].

We can naturally define an LCA $\mathbb{C}oh$ such that $\mathbf{LinRep} \simeq \mathbf{PER}(\mathbb{C}oh)$. Indeed, coherence spaces trivially have linear type structures and there also exists a *universal type*, from which we obtain an untyped LCA $\mathbb{C}oh$ by a linear variant of the Lietz-Streicher theorem [LS02].

Consequently, the category $\mathbf{LinRep} \simeq \mathbf{PER}(\mathbb{C}oh)$ is a model of ILL. ■

As we have seen in the previous section, any total linear maps are uniformly continuous. We next show that any linear map $F : \boldsymbol{X} \longrightarrow_{lin} \boldsymbol{Y}$ can be enhanced with totality, if we assign suitable totalities on $\boldsymbol{X}$ and $\boldsymbol{Y}$. Recall that for any set $\mathcal{A} \subseteq \boldsymbol{X}$ of cliques, its bi-orthogonal $\mathcal{A}^{\perp\perp}$ can be a totality on $\boldsymbol{X}$, hence is endowed with a bounded uniformity. Here is an extension lemma for the bi-orthogonal totalities:

Lemma 1. *Let $\mathcal{A} \subseteq \boldsymbol{X}$ and $\mathcal{B} \subseteq \boldsymbol{Y}$ be arbitrary (non-empty) sets of cliques. If $F : \boldsymbol{X} \longrightarrow_{lin} \boldsymbol{Y}$ satisfies $F[\mathcal{A}] \subseteq \mathcal{B}$ then F is indeed total: $F[\mathcal{A}^{\perp\perp}] \subseteq \mathcal{B}^{\perp\perp}$.*

The condition $F[\mathcal{A}] \subseteq \mathcal{B}$ is exactly what we encounter in defining a realizable function tracked by F. For any coherent representation $\boldsymbol{X} \xrightarrow{\delta_{\boldsymbol{X}}} S$, let us endow $\boldsymbol{X}$ with the totality $\mathcal{T}_{\boldsymbol{X}} := \mathsf{dom}(\delta_{\boldsymbol{X}})^{\perp\perp}$. From the above lemma, $f : S \to T$ is linearly realizable if and only if it is tracked by a *total* linear map $F : \boldsymbol{X} \longrightarrow_{lin} \boldsymbol{Y}$. So one can say that a linearly realizable function is in fact a "totally linearly realizable" function.

One can then observe that the standard representation $\boldsymbol{B}_{\mathbb{X}} \xrightarrow{\delta_{\mathbb{X}}} \mathbb{X}$ of a separable metrizable space $\mathbb{X}$ is also topologically meaningful for linear realizability, like admissible representations for stable realizability:

Theorem 16. *Let $\mathbb{X}$ and $\mathbb{Y}$ be separable metrizable spaces represented by the standard representations. Then every uniformly continuous function $f : \mathbb{X} \to \mathbb{Y}$ is linearly realizable.*

To obtain the converse of Theorem 16, we need to assume on $\mathbb{X}$ a weaker notion of connectedness. A uniform space $\mathbb{X} = (X, \mu_X)$ is *chain-connected* (or sometimes called *uniformly connected*) if given any two points $p, q \in X$ and any uniform cover $\mathcal{U} \in \mu_X$, there exist finitely many $U_1, \ldots U_n \in \mathcal{U}$ such that $p \in U_1$, $U_i \cap U_{i+1} \neq \varnothing$ for every $i < n$, and $U_n \ni q$. Ordinary connectedness implies chain-connectedness. For instance, the set of rational numbers $\mathbb{Q}$ endowed with the usual metric is chain-connected, while it is totally disconnected.

Theorem 17. *Let $\mathbb{X}$ and $\mathbb{Y}$ be separable metrizable spaces represented by the standard representations. Provided that $\mathbb{X}$ is chain-connected, a function $f : \mathbb{X} \to \mathbb{Y}$ is linearly realizable iff it is uniformly continuous.*

Proof Sketch. The "if"-direction is due to Theorem 16. We shall show the "only-if" direction. As noted above, if f is linearly realizable, there exists a total linear map $F : \boldsymbol{B}_{\mathbb{X}} \longrightarrow_{lin} \boldsymbol{B}_{\mathbb{Y}}$ which tracks f, hence F is uniformly continuous w.r.t. the bounded uniformities by Theorem 9. Any standard representation $\delta_{\mathbb{Y}} :\subseteq \boldsymbol{B}_{\mathbb{Y}} \to \mathbb{Y}$ is uniformly continuous as a partial map, so is the composition $\delta_{\mathbb{Y}} \circ F : \mathsf{dom}(\delta_{\mathbb{X}}) \to \mathbb{Y}$. One can show that $\delta_{\mathbb{X}}$ is a uniform quotient by chain-connectedness of $\mathbb{X}$, hence uniform continuity of $f \circ \delta_{\mathbb{X}} = \delta_{\mathbb{Y}} \circ F$ implies that of f. ■

A complete proof is given in the full version.

This result substantially generalizes the already mentioned result ("linear realizability $\Leftrightarrow$ uniform continuity" for real functions) in [MT16].

5 Related and Future Work

Type theory. In this paper, we have proposed coherence spaces with totality as an extension of ordinary coherence spaces, following the idea of Kristiansen and Normann. Originally in domain theory, *domains with totality* are introduced by Berger [Be93] to interpret Martin-Löf type theory using "total" elements. Since our model of coherence spaces with totality is a linear version of this model, one can expect that it could model *intuitionistic linear type theory*.

Our theory also includes a natural representation of (separable metrizable) uniform spaces and uniformly continuous maps between them. Hence it might lead to a denotational model of real functional programming languages (e.g., [Es96, ES14]) extended with other uniform spaces, where one can deal with uniformly continuous functions based on linear types.

Realizability theory. In the traditional setting, giving representations roughly amounts to constructing modest sets over a PCA in the theory of realizability. Our model of coherent representations and stable realizability is in fact considered as a modest set model over a PCA $\mathbb{C}oh$ constructed from coherence spaces, which is known to be a model of intuitionistic logic [Lon94, Ba00]. Bauer then gave an attractive paradigm [Ba05]: "Computable mathematics is a realizability interpretation of constructive mathematics."

On the other hand, less is known about the relationship between computable mathematics and linear realizability theory over a LCA, for which we can build models of *intuitionistic linear logic*. Since we can also construct a LCA from coherence spaces, it is in principle possible to develop such a theory based on our framework. We believe that exploring this direction, already mentioned in [Ba00], will be an interesting avenue for future work.

Acknowledgement. The author is greatful to Naohiko Hoshino and Kazushige Terui (RIMS) for useful comments, and to the anonymous referees for their thoughtful reviews that help improve the manuscript.

This work was partly supported by JSPS Core-to-Core Program (A. Advanced Research Networks) and by KAKENHI 25330013.

References

[AHS02] Abramsky, S., Haghverdi, E., Scott, P.J.: Geometry of interaction and linear combinatory algebras. Math. Struct. in Comput. Sci. **12**(5), 625–665 (2002)

[AL00] Abramsky, S., Lenisa, M.: A fully complete PER model for ML polymorphic types. In: Clote, P.G., Schwichtenberg, H. (eds.) CSL 2000. LNCS, vol. 1862, pp. 140–155. Springer, Heidelberg (2000). doi:10.1007/3-540-44622-2_9

[AL05] Abramsky, S., Lenisa, M.: Linear realizability and full completeness for typed lambda-calculi. Ann. Pure Appl. Logic **134**(2–3), 122–168 (2005)

[As90] Asperti, A.: Stability and computability in coherent domains. Inf. Comput. **86**, 115–139 (1990)

[Ba00] Bauer, A.: The Realizability Approach to Computable Analysis and Topology. Ph.D. thesis, School of Computer Science, Carnegie Mellon University (2000)

[Ba02] Bauer, A.: A relationship between equilogical spaces and type two effectivity. Math. Logic Q. **48**(S1), 1–15 (2002)

[Ba05] Bauer, A.: Realizability as the connection between computable and constructive mathematics. In: Proceedings of CCA , Kyoto, Japan (2005)

[Be93] Berger, U.: Total sets and objects in domain theory. Ann. Pure Appl. Logic **60**, 91–117 (1993)

[BHW08] Brattka, V., Hertling, P., Weihrauch, K.: A tutorial on computable analysis. In: Cooper, S.B., Löwe, B., Sorbi, A. (eds.) New Computational Paradigms, pp. 425–491. Springer, Heidelberg (2008)

[Bi94] Bierman, G.: On intuitionistic linear logic. Ph.D. thesis, University of Cambridge (1994)

[Bl97] Blanck, J.: Computability on topological spaces by effective domain representations. Ph.D. thesis, Uppsala University (1997)

[Ehr05] Ehrhard, T.: Finiteness spaces. Math. Str. Comput. Sci. **15**(4), 615–646 (2005)

[Es96] Escardo, M.H.: PCF extended with real numbers. Theoret. Comput. Sci. **162**(1), 79–115 (1996)

[ES99] Edalat, A., Sunderhauf, P.: A domain-theoretic approach to computability on the real line. Theoret. Comput. Sci. **210**(1), 73–98 (1999)

[ES14] Escardo, M.H., Simpson, A.: Abstract datatypes for real numbers in type theory. In: Proceedings of RTA-TLCA, pp. 208–223 (2014)

[Gi86] Girard, J.-Y.: The system F of variable types, fifteen years later. Theoret. Comput. Sci. **45**(2), 159–192 (1986)

[Gi87] Girard, J.-Y.: Linear logic. Theor. Comput. Sci. **50**(1), 1–101 (1987)

[HS03] Hyland, M., Schalk, A.: Glueing and orthogonality for models of linear logic. Theo. Comput. Sci. **294**(1–2), 183–231 (2003)

[Is64] Isbell, J.R.: Uniform Spaces. American Mathematical Society, Providence (1964)

[Ja99] Jacobs, B.: Categorical Logic and Type Theory. North Holland, Amsterdam (1999)

[Ke75] Kelley, J.L.: General Topology. Springer Science & Business Media, New York (1975)

[KN97] Kristiansen, L., Normann, D.: Total objects in inductively defined types. Arch. Math. Logic **36**, 405–436 (1997)

[Ko91] Ko, K.: Complexity Theory of Real Functions. Birkhäuser, Boston (1991)

[KW85] Kreitz, C., Weihrauch, K.: Theory of representations. Theoret. Comput. Sci. **38**, 35–53 (1985)

[Loa94] Loader, R.: Linear logic, totality and full completeness. In: Proceedings of the 9th Annual IEEE Symposium on Logic in Computer Science, pp. 292–298 (1994)

[Lon94] Longley, J.R.: Realizability Toposes and Language Semantics. Ph.D. thesis, University of Edinburgh (1994)

[LS02] Lietz, P., Streicher, T.: Impredicativity entails untypedness. Math. Struct. Comput. Sci. **12**(3), 335–347 (2002)

[Me09] Mellies, P.-A.: Categorical semantics of linear logic. Interactive models of computation and program behaviour. In: Panoramas et Syntheses vol. 27. Soc. Math. de France (2009)

[MT16] Matsumoto, K., Terui, K.: Coherence spaces for real functions and operators. Submitted (2016). http://www.kurims.kyoto-u.ac.jp/~terui/pub.html

[No90] Normann, D.: Formalizing the notion of total information, pp. 67–94. In: Mathematical Logic. Plenum Press (1990)

[Sc02] Schröder, M.: Extended admissibility. Theoret. Comput. Sci. **284**(2), 519–538 (2002)

[SHT08] Stoltenberg-Hansen, V., Tucker, J.V.: Computability on topological spaces via domain representations. In: Cooper, S.B., Löwe, B., Sorbi, A. (eds.) New Computational Paradigms, pp. 153–194. Springer, New York (2008)

[Si03] Simpson, A.: Towards a category of topological domains. In: Proceedings of the of Thirteenth ALGI Workshop. RIMS, Kyoto University (2003)

[We00] Weihrauch, K.: Computable Analysis – An Introduction. Texts in Theoretical Computer Science. Springer, Heidelberg (2000)

[Wi70] Willard, S.: General Topology. Courier Corp., New York (1970)

The Free Exponential Modality of Probabilistic Coherence Spaces

Raphaëlle Crubillé, Thomas Ehrhard, Michele Pagani(✉), and Christine Tasson

IRIF, UMR 8243, Université Paris Diderot, Sorbonne Paris Cité, 75205 Paris, France
pagani@irif.fr

Abstract. Probabilistic coherence spaces yield a model of linear logic and lambda-calculus with a linear algebra flavor. Formulas/types are associated with convex sets of $\mathbb{R}^+$-valued vectors, linear logic proofs with linear functions and λ-terms with entire functions, both mapping the convex set of their domain into the one of their codomain.

Previous results show that this model is particularly precise in describing the observational equivalences between probabilistic functional programs. We prove here that the exponential modality is the free commutative comonad, giving a further mark of canonicity to the model.

1 Introduction

Linear Logic [6] (LL for short) is a resource aware logic whose relevance in the study of the semantics of computation has been illustrated by several papers. It has three kinds of connectives: additive, mutliplicative and exponential. The connectives of the first two classes are determined by their logical rules: for instance, if one introduces a second tensor product $\otimes'$ in the system, it is easy to prove that $A \otimes B$ and $A \otimes' B$ are canonically equivalent (isomorphic, actually) for all formulas A and B. The exponential connectives behave quite differently: one can add a second exponential modality $!'$ with the same rules as for $!$ and it is not possible to prove that $!A$ and $!'A$ are equivalent in general.

This discrepancy is also well visible at the semantical level. Given a category, the multiplicative-additive structures (if exist) are univocally characterized by universal properties as soon as a notion of multilinear map is given, while various different notions of the exponential modality are possible, in general. Well-known examples are the category **Coh** of coherence spaces, where the exponential modality can be expressed by both the finite clique and the finite multi-clique functors [7], as well as the category **Rel** of sets and relations, for which [2] introduces a wide family of exponential modalities in addtion to the more standard one based on the finite multiset functor.

Choosing an exponential interpretation in the model is not at all anodyne, as it defines the way we express the structural rules of weakening and contraction, or, from a programming perspective, the operations of erasure and copy.

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 20–35, 2017.
DOI: 10.1007/978-3-662-54458-7_2

For instance, the new exponential modalities in **Rel**, introduced by [2], allow to construct non-sensible models of the untyped λ-calculus, while the models based on the finite multiset functor are all sensible. Let us also mention that this variety of exponentials has been used for modeling different notions of co-effects [1].

Any interpretation of the ! LL modality is an operation transforming an object A into a commutative comonoid $!A$ with respect to $\otimes$, the comultiplication allowing one to interpret contraction, and the coneutral element allowing one to interpret weakening. This simple fact shows that among all possible notions of !, there might be one which is more canonical than any other, being the terminal object in the category of commutative comonoids over A. Lafont [9] proved in fact that the free commutative comonoid, whenever it exists, always gives an interpretation of the exponential modality, which is then called the *free exponential modality*, and written as $!_f$.

The study of this free exponential $!_f A$, when it exists, is particularly important because its universal property expresses a direct connection with the tensor product of the categorical model of LL under consideration. This fact is particularly well illustrated by [11] where it is shown that, under some fairly general assumptions, this free exponential can be described as a kind of "symmetric infinite tensor product"; we come back to this characterization later since it is at the core of the present work.

The free exponential modality is well-kwown in both **Coh** and **Rel**, being given respectively by the multi-clique and the multi-set functors. The goal of this paper is to prove the existence of and to study the free exponential modality of the category **Pcoh** of probabilistic coherence spaces.

Pcoh is a model of linear logic carrying both linear algebraic and probabilistic intuitions. Sketched by Girard in [8], it was more deeply investigated in [3]. Formulas/types are interpreted as suitable sets $\mathrm{P}(\mathcal{A})$, $\mathrm{P}(\mathcal{B})$,... of real-valued vectors which are both Scott-closed (allowing the usual definition of fixed-points) and convex (i.e. closed under barycentric sums). This latter feature makes **Pcoh** particularly suitable for modeling (discrete) probabilistic computations: the barycentric sum $\kappa v + \rho w$ (for $\kappa, \rho \in [0,1]$, $\kappa + \rho \leq 1$, v, w vectors) expresses a computation which returns v with probability κ, w with probability ρ, and the remainder $1 - (\kappa + \rho)$ is the probability that the computation diverges.

The morphisms of **Pcoh** are Scott-continuous and linear functions mapping the set $\mathrm{P}(\mathcal{A})$ of vectors associated with the domain into the set $\mathrm{P}(\mathcal{B})$ associated with the codomain. These linear maps are described as (usually infinite-dimensional) matrices and the set of these matrices is the hom-object $\mathrm{P}(\mathcal{A} \multimap \mathcal{B})$ associated with linear implication. From a programming perspective, the matrices in $\mathrm{P}(\mathcal{A} \multimap \mathcal{B})$ correspond to probabilistic programs that use their inputs exactly once in order to compute a result. In order to interpret non-linear programs one has to choose an exponential modality. The only known ! modality of **Pcoh** is the one introduced in [3], which is based on the notion of entire function and hence we will call it the *entire exponential modality* and denote it with $!_e$. The set $\mathrm{P}(!_e\mathcal{A} \multimap \mathcal{B})$ can be seen as a set of matrices giving the coefficients of a power series expressing an entire function from $\mathrm{P}(\mathcal{A})$ to $\mathrm{P}(\mathcal{B})$. As usual the interpretation of a (call-by-name) program of type $A \Rightarrow B$ is represented as an LL proof of $!A \multimap B$ and hence as an entire function in **Pcoh**.

The entire exponential modality of **Pcoh** has been recently shown to be particularly relevant for describing the observational behavior of higher-order probabilistic programs. Namely, [4] proves that the Kleisli category associated with $!_e$ is fully-abstract with respect to call-by-name probabilistic PCF, while [5] proves, using the Eilenberg-Moore category, that **Pcoh** is fully-abstract for a call-by-push-value version of probabilistic PCF. Let us stress that these are indeed the only known fully-abstract models for probabilistic PCF.

It is then natural to ask whether $!_f$ exists in **Pcoh** and if it is the case, how it relates with $!_e$. In this paper we answer the first question positively and prove that $!_f$ and $!_e$ are the same modality, in spite of their different presentations.

Our main tool is a recipe for constructing $!_f$ out of a model of the multiplicative additive fragment of LL, given by Melliès, Tabareau and Tasson [11]. The idea is to adapt the well-known formula defining the symmetric algebra generated by a vector space, the latter being the free commutative monoid. More precisely, the recipe gives (under suitable conditions) $!_f\mathcal{A}$ as the limit of a sequence of approximants $\mathcal{A}^{\leq n}$ which correspond to the equalizers of the tensor symmetries of a suitable space (see Sect. 3). At a first sight, $\mathcal{A}^{\leq n}$ moves us far away from the entire exponential $!_e\mathcal{A}$ of [3], in fact the coefficients appearing in $\mathcal{A}^{\leq n}$ are greater than those of $!_e\mathcal{A}$, so that one is tempted to suppose that $!_f\mathcal{A}$ is a space much bigger than $!_e\mathcal{A}$, exactly as it is the case for standard coherence spaces where the images under the finite multi-clique functor strictly contain those under the finite clique functor (Example 5). We prove that this is not the case, the limit $!_f\mathcal{A}$ of the approximants $\mathcal{A}^{\leq n}$ exists (Proposition 4), and, more surprisingly, it is equal to the entire exponential modality (Proposition 5). For any n, the coefficients of $\mathcal{A}^{\leq n}$ are larger than those of $!_e\mathcal{A}$, but as $n \to \infty$, these coefficients tend to be exactly those of $!_e\mathcal{A}$ (see Sect. 4.3).

Contents of the paper. In Sect. 2, we briefly recall the semantics of LL in **Pcoh** as defined in [3]. In particular, Eqs. (8) and (9) sketch the definition of the entire exponential modality. Section 3 gives the categorical definition of the free exponential modality and Melliés, Tabareau and Tasson's recipe for constructing it (Proposition 2). The major contributions are in Sect. 4, where we apply this recipe to **Pcoh** (Propositions 3 and 4) and prove that the free exponential modality is equal to the entire one (Proposition 5).

Notation. We write as S_n the set of permutations of the set $\{1, \dots, n\}$, the factorial $n!$ being its cardinality. The writing $f : A \hookrightarrow B$ denotes an injection f from a set A into a set B. A multiset μ over A is a function from A to $\mathbb{N}$, $\mathrm{Supp}(\mu)$ denoting its support $\{a \in A \ ; \ \mu(a) \neq 0\}$, and $\#\mu$ its cardinality $\sum_{a \in A} \mu(a)$. Given $n \in \mathbb{N}$, $\mathcal{M}_n(A)$ is the set of multisets of cardinality n, while $\mathcal{M}_{\mathrm{f}}(A)$ is the set of finite multisets, i.e. $\mathcal{M}_{\mathrm{f}}(A) = \bigcup_n \mathcal{M}_n(A)$. A multiset μ can also be presented by listing the occurrences of its elements within brackets, like $\mu = [a, a, b]$, as well as $\mu = [a^2, b]$, for the multiset $\mu(a) = 2$, $\mu(b) = 1$, $\mathrm{Supp}(\mu) = \{a, b\}$. Finally, $\mu \uplus \nu$ is the disjoint union of two multisets, i.e. $(\mu \uplus \nu)(a) = \mu(a) + \nu(a)$, the empty multiset $[\,]$ being its neutral element. We denote by $\boldsymbol{a}$ a finite sequence of elements of A, $\boldsymbol{a}_i$ being its i-th element.

Given such a sequence $\boldsymbol{a}$, we denote as $\widetilde{\boldsymbol{a}}$ the multiset of its elements, and given a multiset μ, we write as $\mathrm{Enum}(\mu)$ the set of all its different enumerations. The cardinality of $\mathrm{Enum}(\mu)$ depends on μ and can be given by the *multinomial coefficient* $m(\mu)$:

$$\widetilde{\boldsymbol{a}}=[a_1,\dots,a_n], \qquad \text{for } \boldsymbol{a}=(a_1,\dots,a_n), \tag{1}$$

$$\mathrm{Enum}(\mu)=\{(a_{\sigma(1)},\dots,a_{\sigma(n)})\ ;\ \sigma\in S_n\}, \qquad \text{for } \mu=[a_1,\dots,a_n], \tag{2}$$

$$m(\mu)=\frac{\#\mu!}{\prod_{a\in\mathrm{Supp}(\mu)}\mu(a)!}. \tag{3}$$

2 The Model of Probabilistic Coherence Spaces

In order to be self-contained, we shortly recall the linear logic model based on probabilistic coherence spaces, as defined in [3]. For the sake of brevity we will omit the proofs of this section.

Let I be a set, for any vectors $v,w\in(\mathbb{R}^+)^I$, the pairing is defined as usual

$$\langle v,w\rangle=\sum_{i\in I}v_i\,w_i\in\mathbb{R}^+\cup\{\infty\}. \tag{4}$$

Given a set $\mathrm{P}\subseteq(\mathbb{R}^+)^I$ we define $\mathrm{P}^\perp$, the *polar* of P, as

$$\mathrm{P}^\perp=\{w\in(\mathbb{R}^+)^I\mid\forall v\in\mathrm{P}\ \ \langle v,w\rangle\leq 1\}. \tag{5}$$

The polar satisfies the following immediate properties: $\mathrm{P}\subseteq\mathrm{P}^{\perp\perp}$, if $\mathrm{P}\subseteq\mathrm{Q}$ then $\mathrm{Q}^\perp\subseteq\mathrm{P}^\perp$, and then $\mathrm{P}^\perp=\mathrm{P}^{\perp\perp\perp}$.

Definition 1 ([3,8]). *A* probabilistic coherence space*, or PCS for short, is a pair* $\mathcal{A}=(|\mathcal{A}|,\mathrm{P}(\mathcal{A}))$ *where* $|\mathcal{A}|$ *is a countable set called the* web *of* $\mathcal{A}$ *and* $\mathrm{P}(\mathcal{A})$ *is a subset of* $(\mathbb{R}^+)^{|\mathcal{A}|}$ *such that the following holds:*

closedness: $\mathrm{P}(\mathcal{A})^{\perp\perp}=\mathrm{P}(\mathcal{A})$,
boundedness: $\forall a\in|\mathcal{A}|,\ \exists\mu>0,\ \forall v\in\mathrm{P}(\mathcal{A}),\ v_a\leq\mu$,
completeness: $\forall a\in|\mathcal{A}|,\ \exists\lambda>0,\ \lambda e_a\in\mathrm{P}(\mathcal{A})$,

where e_a *denotes the* base vector *for* a*:* $(e_a)_b=\delta_{a,b}$*, with* δ *the Kronecker delta.*

The dual of a PCS $\mathcal{A}$ *is defined by* $\mathcal{A}^\perp=(|\mathcal{A}|,\mathrm{P}(\mathcal{A})^\perp)$.

Notice that we do not require $\mathrm{P}(\mathcal{A})\subseteq[0,1]^{|\mathcal{A}|}$, we shall understand why with the exponential construction (see Example 4).

We consider $(\mathbb{R}^+\cup\{\infty\})^{|\mathcal{A}|}$ endowed with the pointwise order:

$$v\leq w=\forall a\in|\mathcal{A}|,\ v_a\leq w_a, \tag{6}$$

with the lub of $P\subseteq(\mathbb{R}^+\cup\{\infty\})^{|\mathcal{A}|}$ given by: $\forall a\in|\mathcal{A}|$, $(\sup P)_a=\sup_{v\in P}v_a$. Notice that, thanks to boundedness, $(\sup\mathrm{P}(\mathcal{A}))_a\in\mathbb{R}^+$, we denote it by $(\sup\mathcal{A})_a$ and call it the *greatest coefficient of* $\mathcal{A}$ *along* a.

The following is a variant of a theorem in [8], giving an equivalent presentation of a PCS, based on the notions of Scott closure and convex closure. Its proof is a standard application of the Hahn-Banach Theorem.

Proposition 1 ([8]). *Let I be a countable set and $S \subseteq \mathbb{R}^{+I}$. The pair (I, S) is a probabilistic coherence space iff:*

1. *S is bounded and complete (see Definition 1);*
2. *S is* Scott closed, *i.e. $\forall u \leq v \in S$, $u \in S$, and $\forall D \subseteq S$ directed, $\sup D \in S$;*
3. *S is* convex, *i.e. $\forall u, v \in S$, $\lambda \leq 1$, $\lambda u + (1-\lambda)v \in S$.*

Definition 2. *The category* **Pcoh** *has as* objects *PCSs and the set* $\mathbf{Pcoh}(\mathcal{A}, \mathcal{B})$ *of* morphisms *from $\mathcal{A}$ to $\mathcal{B}$ is the set of those matrices $f \in (\mathbb{R}^+)^{|\mathcal{A}|\times|\mathcal{B}|}$ s.t.:*

$$\forall v \in \mathrm{P}(\mathcal{A}), f\,v \in \mathrm{P}(\mathcal{B}), \tag{7}$$

where $f\,v$ is the usual matricial product: $\forall b \in |\mathcal{B}|$, $(f\,v)_b = \sum_{a\in|\mathcal{A}|} f_{a,b} v_a$.

The identity $\mathrm{id}^{\mathcal{A}}$ *on $\mathcal{A}$ is defined as the diagonal matrix given by* $(\mathrm{id}^{\mathcal{A}})_{a,a'} = \delta_{a,a'}$. Composition *of morphisms is matrix multiplication:* $(g \circ f)_{a,c} = \sum_{b\in|\mathcal{B}|} f_{a,b} g_{b,c}$, *where $f \in \mathbf{Pcoh}(\mathcal{A}, \mathcal{B})$, $g \in \mathbf{Pcoh}(\mathcal{B}, \mathcal{C})$, and $a \in |\mathcal{A}|$, $c \in |\mathcal{C}|$.*

The above sum $\sum_{b\in|\mathcal{B}|} f_{a,b} g_{b,c}$ converges in $\mathbb{R}^+$, because f, g enjoy condition (7).

We will often use Lemma 1, allowing us to infer condition (7) for all $v \in \mathrm{P}(\mathcal{A})$, just by testing it on a set G of generators of $\mathrm{P}(\mathcal{A})$ (see the proof in Appendix):

Lemma 1. *Let $\mathcal{A}, \mathcal{B}$ be two probabilistic coherence spaces and f be a matrix in $\mathbb{R}^{+|\mathcal{A}|\times|\mathcal{B}|}$. Let $G \subseteq \mathrm{P}(\mathcal{A})$ such that: (i) $G^{\perp\perp} = \mathrm{P}(\mathcal{A})$, and (ii) $\forall v \in G$, $f\,v \in \mathbb{R}^{+|\mathcal{B}|}$, then: $f(G)^{\perp\perp} = f(\mathrm{P}(\mathcal{A}))^{\perp\perp}$.*

Monoidal Structure. The bifunctor $\otimes : \mathbf{Pcoh} \times \mathbf{Pcoh} \to \mathbf{Pcoh}$ is defined as

$$|\mathcal{A} \otimes \mathcal{B}| = |\mathcal{A}| \times |\mathcal{B}|, \qquad \mathrm{P}(\mathcal{A} \otimes \mathcal{B}) = \{v \otimes w \,;\, v \in \mathrm{P}(\mathcal{A}), w \in \mathrm{P}(\mathcal{B})\}^{\perp\perp},$$

where $(x \otimes y)_{a,b} = x_a y_b$, for $a \in |\mathcal{A}|$ and $b \in |\mathcal{B}|$. The action of $\otimes$ on morphisms $u \in \mathbf{Pcoh}(\mathcal{A}, \mathcal{B})$ and $v \in \mathbf{Pcoh}(\mathcal{A}', \mathcal{B}')$ is defined as $(u \otimes v)_{(a,a'),(b,b')} = u_{a,b} v_{a',b'}$, for $(a, a') \in |\mathcal{A} \otimes \mathcal{A}'|$, $(b, b') \in |\mathcal{B} \otimes \mathcal{B}'|$. The symmetry $\mathtt{swap} \in \mathbf{Pcoh}(\mathcal{A} \otimes \mathcal{B}, \mathcal{B} \otimes \mathcal{A})$ is given by $\mathtt{swap}_{(a,b),(b',a')} = \delta_{a,a'}\delta_{b,b'}$. The other natural isomorphisms (associativity, neutrality) are given similarly.

In the next sections, we will often refer to the n-fold ($n \in \mathbb{N}$) tensor product over a PCS $\mathcal{A}$, which can be presented as: $|\mathcal{A}^{\otimes n}| = \{(a_1, \ldots, a_n) \,;\, a_i \in |\mathcal{A}|\}$, and $\mathrm{P}(\mathcal{A}^{\otimes n}) = \{v_1 \otimes \cdots \otimes v_n \,;\, v_i \in \mathrm{P}(\mathcal{A})\}^{\perp\perp}$. Notice that any permutation $\sigma \in S_n$ defines a symmetry over $\mathcal{A}^{\otimes n}$, which we denote in the same way: $\sigma_{(a_1,\ldots,a_n),(a'_1,\ldots,a'_n)} = \prod_{i=1}^{n} \delta_{a_i, a'_{\sigma(i)}}$. For example, the image of a vector of the form $v_1 \otimes \cdots \otimes v_n$ under σ is $v_{\sigma(1)} \otimes \cdots \otimes v_{\sigma(n)}$. The unit of $\otimes$ is given by the singleton web PCS $\mathbf{1} = (\{\star\}, [0,1]^{\{\star\}})$, which is also equal to $\mathcal{A}^{\otimes 0}$ for any $\mathcal{A}$.

The object of linear morphisms $\mathcal{A} \multimap \mathcal{B}$ is defined as

$$|\mathcal{A} \multimap \mathcal{B}| = |\mathcal{A}| \times |\mathcal{B}|, \qquad \mathrm{P}(\mathcal{A} \multimap \mathcal{B}) = \mathbf{Pcoh}(\mathcal{A}, \mathcal{B}).$$

Pcoh is $\star$-autonomous, the dualizing object $\perp$ being defined as the dual of $\mathbf{1}$ which is indeed equal to $\mathbf{1}$: $\perp = \mathbf{1}^{\perp} = \mathbf{1}$.

Cartesian Structure. Pcoh admits cartesian products of any countable family $(\mathcal{A}_i)_{i\in I}$ of PCSs, defined as

$$|\&_{i\in I}\mathcal{A}_i| = \bigcup_{i\in I}(\{i\}\times|\mathcal{A}_i|), \quad \mathrm{P}(\&_{i\in I}\mathcal{A}_i) = \left\{v\in\mathbb{R}^{+|\&_{i\in I}\mathcal{A}_i|}; \forall i\in I, \pi_i\, v\in\mathrm{P}(\mathcal{A}_i)\right\}$$

where $\pi_i\, v$ is the vector in $(\mathbb{R}^+)^{|\mathcal{A}_i|}$ denoting the i-th component of v, i.e. $\pi_i\, v_a = v_{(i,a)}$. The j-th projection $\mathrm{pr}^j \in \mathbf{Pcoh}(\&_{i\in I}\mathcal{A}_i, \mathcal{A}_j)$ is $\mathrm{pr}^j_{(i,a),b} = \delta_{i,j}\delta_{a,b}$.

Notice that the empty product yields the terminal object $\top$ defined as $|\top| = \emptyset$ and $\top = \{\mathbf{0}\}$. We may write the binary product by $\mathcal{A}_1 \,\&\, \mathcal{A}_2$, as well as any $v\in\mathrm{P}(\mathcal{A}_1 \,\&\, \mathcal{A}_2)$ by the pair $(\pi_1\, v, \pi_2\, v)\in\mathrm{P}(\mathcal{A}_1)\times\mathrm{P}(\mathcal{A}_2)$ of its components.

Example 1. All examples in this paper will be built on top of the flat interpretation of the boolean type, which is defined as the space $Bool = (\bot \,\&\, \bot)^\bot$, which can be equally written as $\mathbf{1}\oplus\mathbf{1}$, with $\oplus$ referring to the co-product. The web $|Bool|$ has two elements, that we denote as $\mathtt{t}$ and $\mathtt{f}$. The set $\mathrm{P}(Bool)$ is $\{\kappa e_{\mathtt{t}} + \rho e_{\mathtt{f}}\ ;\ \kappa+\rho\leq 1\}$, that is the sub-probabilistic distributions on the base vectors $e_{\mathtt{t}}$, $e_{\mathtt{f}}$. This is because $\mathrm{P}(Bool)^\bot = \mathrm{P}(\bot\,\&\,\bot) = \{\kappa e_{\mathtt{t}} + \rho e_{\mathtt{f}}\ ;\ \kappa,\rho\leq 1\}$.

Example 2. Let us compute $Bool^{\otimes 2} = Bool\otimes Bool$. By definition we have: $\left|Bool^{\otimes 2}\right| = \{(\mathtt{t},\mathtt{t}),(\mathtt{t},\mathtt{f}),(\mathtt{f},\mathtt{t}),(\mathtt{f},\mathtt{f})\}$, $\mathrm{P}\left(Bool^{\otimes 2}\right) = \{v\otimes w\ ;\ v,w\in\mathrm{P}(Bool)\}^{\bot\bot}$. Using Example 1, one checks that $\{v\otimes w\ ;\ v,w\in\mathrm{P}(Bool)\}^\bot$ is equal to $\{u\in\mathbb{R}^{+\left|Bool^{\otimes 2}\right|}\ ;\ \forall(b,b')\in\left|Bool^{\otimes 2}\right|, u_{b,b'}\leq 1\}$. Hence, $\mathrm{P}\left(Bool^{\otimes 2}\right)$ is the set $\{u\ ;\ \sum_{(b,b')\in\{\mathtt{t},\mathtt{f}\}^2} u_{b,b'}\leq 1\}$ of sub-probability distributions of pairs of booleans.

Example 3. As for $Bool\multimap Bool$, its web is $|Bool\multimap Bool| = |Bool\otimes Bool|$, and $f\in\mathrm{P}(Bool\multimap Bool)$ whenever $\forall v\in\mathrm{P}(Bool)$, $f\,v\in\mathrm{P}(Bool)$. By Example 1, this is equivalent to $\forall\kappa+\rho\leq 1$, $f(\kappa e_{\mathtt{t}} + \rho e_{\mathtt{f}})_{\mathtt{t}} + f(\kappa e_{\mathtt{t}} + \rho e_{\mathtt{f}})_{\mathtt{f}}\leq 1$. By linearity, the condition boils down to $f_{\mathtt{t},\mathtt{t}} + f_{\mathtt{t},\mathtt{f}}\leq 1$ and $f_{\mathtt{f},\mathtt{t}} + f_{\mathtt{f},\mathtt{f}}\leq 1$. That is: $\mathrm{P}(Bool\multimap Bool)$ is the set of stochastic matrices over $\{\mathtt{t},\mathtt{f}\}$, as expected.

Exponential Structure. We recall the exponential modality given in [3] and we call it the *entire exponential modality*, denoted by $!_e$, in opposition to the *free exponential modality* $!_f$ whose definition is the goal of Sect. 4. The main result of the paper is proving that the two modalities, although different in the presentation, actually give the same object (Proposition 5). The adjective *entire* is motivated by the key property that the morphisms from $\mathcal{A}$ to $\mathcal{B}$ in the Kleisli category associated with $!_e$, that is the linear morphisms from $!_e\mathcal{A}$ to $\mathcal{B}$, can be seen as entire functions from $\mathrm{P}(\mathcal{A})$ to $\mathrm{P}(\mathcal{B})$. We refer to [3] for details.

The functorial promotion $!_e : \mathbf{Pcoh}\to\mathbf{Pcoh}$ is defined on objects as

$$|!_e\mathcal{A}| = \mathcal{M}_{\mathrm{f}}(|\mathcal{A}|), \qquad \mathrm{P}(!_e\mathcal{A}) = \{v^{!_e}\ ;\ v\in\mathrm{P}(\mathcal{A})\}^{\bot\bot}, \tag{8}$$

where $v^{!_e}$ is the vector of $(\mathbb{R}^+)^{\mathcal{M}_{\mathrm{f}}(|\mathcal{A}|)}$ defined as $v^{!_e}_{\mu} = \prod_{a \in \mathrm{Supp}(\mu)} v_a^{\mu(a)}$, for any $\mu \in \mathcal{M}_{\mathrm{f}}(|\mathcal{A}|)$. The action of $!_e$ on a morphism $f \in \mathbf{Pcoh}(\mathcal{A}, \mathcal{B})$ is defined, for any $\mu \in |!_e\mathcal{A}|$ and $\nu = [b_1, \ldots, b_n] \in |!_e\mathcal{B}|$, as:

$$(!_e f)_{\mu,\nu} = \sum_{\substack{(a_1,\ldots,a_n)\ ; \\ [a_1,\ldots,a_n]=\mu}} \prod_{i=1}^{n} f_{a_i,b_i} \tag{9}$$

Notice that the above sum varies on the set of *different* enumerations of μ. If μ is a multiset $[a, \ldots, a]$ of support a singleton, then the sum has only one term, while in case of multisets with no repetitions, the sum has $n!$ terms. Remark also that the definition is independent from the chosen enumeration of ν.

Example 4. Equation (9) introduces arbitrary large scalars, moving us away from the intuitive setting of distributions and stochastic matrixes (see Examples 1 and 3). For an example, consider the morphism $f \in \mathbf{Pcoh}(\mathit{Bool}, \mathbf{1})$ defined by $f_{\mathtt{t},*} = f_{\mathtt{f},*} = 1$. Remark that for any $n, m \in \mathbb{N}$, we have: $!_e f_{[\mathtt{t}^n,\mathtt{f}^m],[\star^{n+m}]} = \frac{(n+m)!}{n!m!}$, which is the number of different enumerations of the multiset $[\mathtt{t}^n, \mathtt{f}^m]$ with n (resp. m) occurrences of $\mathtt{t}$ (resp. $\mathtt{f}$). This shows why, in the definition of a PCS, coefficients have to be in the whole of $\mathbb{R}^+$ and cannot be restricted to $[0, 1]$.

The functorial promotion is equipped with a structure of comonad. The counit (or *dereliction*) $\mathtt{der}_{\mathcal{A}} \in \mathbf{Pcoh}(!_e\mathcal{A}, \mathcal{A})$ is defined as $(\mathtt{der}_{\mathcal{A}})_{\mu,a} = \delta_{\mu,[a]}$. The comultiplication (or *digging*), denoted as $\mathtt{digg}_{\mathcal{A}} \in \mathbf{Pcoh}(!_e\mathcal{A}, !_e!_e\mathcal{A})$, is given by $(\mathtt{digg}_{\mathcal{A}})_{\mu,M} = \delta_{\mu,\biguplus M}$, where $\biguplus M$ is the multiset in $|!_e\mathcal{A}|$ obtained as the multiset union of the multisets in $M \in |!_e!_e\mathcal{A}|$.

The PCSs $!_e\mathcal{A} \otimes !_e\mathcal{B}$ and $!_e(\mathcal{A} \,\&\, \mathcal{B})$ are naturally isomorphic, hence we get a model of LL, according to the so-called *new-Seely axiomatisation* (see [10]). In particular, contraction $\mathtt{contr}_{\mathcal{A}} \in \mathbf{Pcoh}(!_f\mathcal{A}, !_f\mathcal{A} \otimes !_f\mathcal{A})$ and weakening $\mathtt{weak}_{\mathcal{A}} \in \mathbf{Pcoh}(!_f\mathcal{A}, \mathbf{1})$ are given by $(\mathtt{contr}_{\mathcal{A}})_{\mu,(\mu',\mu'')} = \delta_{\mu,\mu' \uplus \mu''}$, and $(\mathtt{weak}_{\mathcal{A}})_{\mu,\star} = \delta_{\mu,[]}$.

3 The Free Exponential Modality

We first recall Lafont's axiomatisation of linear logic models (Theorem 1), this is a more restricted axiomatisation than the new-Seely one. The key notion is the free exponential modality. We then recall a general recipe by Melliès, Tasson and Tabareau [11] for constructing this free modality whenever specific conditions hold (Proposition 2). Section 4 applies this recipe to **Pcoh**.

3.1 Lafont's Model

By definition, the structure of an exponential modality turns an object A into a commutative comonoid $!A$, with the comultiplication given by contraction and the neutral element given by weakening: $1 \xleftarrow{\mathtt{weak}_{!A}} !A \xrightarrow{\mathtt{contr}_{!A}} !A \otimes !A$. The converse does not hold in general, since commutative comonoids may lack a

comonad structure (dereliction, digging, and the action on morphisms). However, Lafont proves that in case the commutative comonoid is the free one then its universal property allows one to canonically construct the missing structure:

Theorem 1 ([9]). *A $\star$-autonomous category $\mathbb{C}$ is a model of linear logic if:*

1. *it has finite products and,*
2. *for every object A, there exists a triplet $(!_f A, \mathtt{weak}_{!_f A}, \mathtt{contr}_{!_f A})$ which is the free commutative comonoid generated by A.*

Condition (2) requires the comonoid $!_f A$ to be endowed with a morphism $\mathtt{der}_A \in \mathbb{C}(!_f A, A)$, such that, for all commutative comonoid C and $f \in \mathbb{C}(C, A)$, there exists exactly one commutative comonoid morphism $f^\dagger$ such that $f^\dagger \circ \mathtt{der}_{!_f A} = f$.

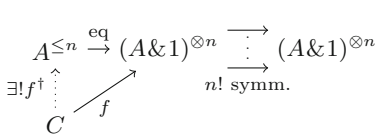
This corresponds to saying that the category of commutative comonoids over A has $!_f A$ as the terminal object.

Example 5 **(Coh).** The first model of LL was introduced by Girard using the notion of coherence space [6]. A coherence space $A = (|A|, \frown_A)$ is a pair of a set $|A|$, the *web*, and a symmetric reflexive relation $\frown_A$, the *coherence*. The exponential modality of the original model is given by the finite cliques functor, $|!A| = \{x \subseteq_f |A| \mid \forall a, a' \in x, a \frown_A a'\}$, while the free exponential modality $|!_f A| = \{\mu \in \mathcal{M}_{\mathrm{f}}(|A|) \mid \forall a, a' \in \mathrm{Supp}(\mu), a \frown_A a'\}$ is given by the finite *multi*-cliques functor, as shown in [12]. The morphism $(\mathtt{der}_{!A})^\dagger$ factoring $!A$ through $!_f A$ is given by the support relation: $(\mathtt{der}_{!A})^\dagger = \{(\mathrm{Supp}(\mu), \mu) \ ; \ \mu \in |!_f A|\}$.

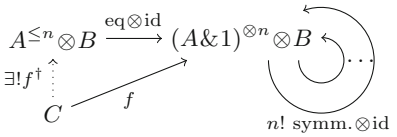
3.2 Melliès, Tasson and Tabareau's Formula

Melliès et al. give a recipe for constructing free commutative comonoids in [11], adapting the well-known formula defining the symmetric algebra generated by a vector space in the setting of LL. This adaptation is non-trivial mainly because the vector space construction uses biproducts, while products and coproducts are in general distinct in LL models, and in fact this is the case for **Pcoh**.

The idea of [11] is to define $!_f A$ as the projective limit of the sequence of its "approximants" $A^{\leq 0}, A^{\leq 1}, A^{\leq 2}, \ldots$, where an approximant $A^{\leq n}$ is the equalizer of the $n!$ tensor symmetries over $(A \,\&\, 1)^{\otimes n}$. Intuitively, the object $A^{\leq n}$ describes the behavior of data of type $!_f A$ when duplicated *at most* n times. This behavior is given by the equalizers of the tensor symmetries because the model does not distinguish between the evaluation order of the n copies (in categorical terms, we are considering *commutative* comonoids). We start from $A \,\&\, 1$ instead of A because, in order to have a sequence, we need that each $A^{\leq n}$ in some sense encompasses its predecessors $A^{\leq 0}, A^{\leq 1}, \ldots, A^{\leq n-1}$. The exact meaning of "*to encompass*" is given by a family of morphisms $\rho_{n,n-1} \in \mathbb{C}(A^{\leq n} \mapsto A^{\leq n-1})$ generated by the right projection of the product $A \,\&\, 1$ (see Notation 2). In standard coherence spaces, this turns out to be the simple fact that the set of cliques of $A^{\leq n+1}$ *contains* the cliques of $A^{\leq n}$. In contrast, this intuition is misleading for **Pcoh** (Example 8), making our construction considerably subtler.



(a) Universal property of $A^{\leq n}$, eq: $\forall C, \forall f \in \mathbb{C}(C, (A \otimes 1)^{\otimes n})$ invariant under $\otimes$ symm., $\exists! f^\dagger \in \mathbb{C}(C, A^{\leq n})$ commuting the diagram.



(b) diagram defining the commutation of $A^{\leq n}$ with the $\otimes$ product.

Fig. 1. Properties of the approximants $A^{\leq n}$ of $!_f A$

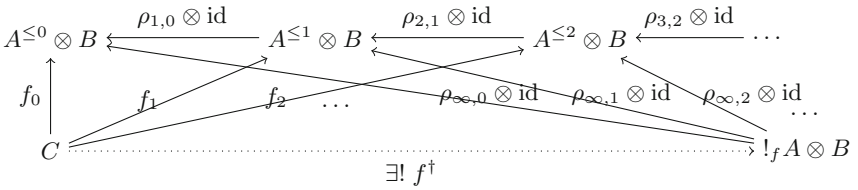


Fig. 2. Diagram defining the commutation of the limit $!_f A$ with $\otimes$: for every objects B, C and family of morphisms $(f_i)_{i \in \mathbb{N}}$ making the diagram above commute, there exists a unique $f^\dagger \in \mathbb{C}(C, !_f A \otimes B)$ making the diagram commute.

Notation 2. *Given an object A of a symmetric monoidal category with finite cartesian products $\mathbb{C}$ and a number $n \in \mathbb{N}$, we denote by $A^{\leq n}$ the equalizer of the $n!$ $\otimes$ symmetries of $(A \,\&\, 1)^{\otimes n}$ (Fig. 1a), whenever it exists. Moreover, we denote by $\rho_{n+1,n}$ the morphism $\mathbb{C}(A^{\leq n+1}, A^{\leq n})$ obtained by applying the universal property of $A^{\leq n}$ to $(\mathtt{id} \otimes \mathrm{pr}_r) \circ \mathtt{eq}$, i.e. taking in Fig. 1a the object $C = A^{\leq n+1}$ and $f = (\mathtt{id} \otimes \mathrm{pr}_r) \circ \mathtt{eq}$, with pr_r denoting the right projection of $A \,\&\, 1$.*

Proposition 2. **([11]).** *Let A be an object of a cartesian symmetric monoidal category $\mathbb{C}$, the free commutative comonoid generated by A is the limit of:*

$$A^{\leq 0} \xleftarrow{\rho_{1,0}} A^{\leq 1} \xleftarrow{\rho_{2,1}} A^{\leq 2} \xleftarrow{\rho_{3,2}} A^{\leq 3} \xleftarrow{\rho_{4,3}} \cdots$$

provided that:

1. *$\forall n \in \mathbb{N}$, the equalizer $A^{\leq n}$ exists and commutes with tensor products (Fig. 1b);*
2. *the limit $!_f A$ of the diagram exists and commutes with tensor products (Fig. 2).*

4 The Case of Probabilistic Coherence Spaces

In this section we prove that **Pcoh** is a Lafont model. First, we define the free commutative comonad $!_f \mathcal{A}$ (Definition 3 and Corollary 1) by applying Melliès et al.'s recipe (Subsects. 4.1 and 4.2), then we prove that $!_f \mathcal{A}$ is actually equivalent to the entire exponential modality $!_e \mathcal{A}$ defined in Sect. 2 (Proposition 5).

4.1 The Approximants $\mathcal{A}^{\leq N}$

The first step for applying Proposition 2 is to define the approximants $\mathcal{A}^{\leq n}$ and to prove that they commute with the tensor product (Condition 2 of Proposition 2). By definition $\mathcal{A}^{\leq n}$ is the equalizer of the symmetries of $(\mathcal{A} \,\&\, \mathbf{1})^{\otimes n}$, so first we prove that the equalizer $\mathcal{A}^n$ of the symmetries of $\mathcal{A}^{\otimes n}$ exists for any space $\mathcal{A}$ and that it commutes with $\otimes$ (Proposition 3), then we show (Lemma 3) that $\mathcal{A}^{\leq n} = (\mathcal{A} \,\&\, \mathbf{1})^n$ can be directly defined from $\mathcal{A}$, by means of an operator $\langle u_1, \dots u_n \rangle$ over the vectors in $\mathrm{P}(\mathcal{A})$, crucial for the next step.

Recall from Sect. 2 that $\mathcal{A}^{\otimes n}$ can be described as having as web the set of length n sequences $\boldsymbol{a}$ of elements in $|\mathcal{A}|$. Recall also the notation given in Sect. 1. Given the permutation group S_n, we define an endomorphism s_n over $\mathcal{A}^{\otimes n}$, by taking the barycentric sum of the actions of S_n over $\mathcal{A}$, so: $s_n = \frac{1}{n!}\sum_{\sigma \in S_n} \sigma$, which can be explicitly defined as a matrix by: $(s_n)_{\boldsymbol{a},\boldsymbol{a}'} = \delta_{\widetilde{\boldsymbol{a}},\widetilde{\boldsymbol{a}'}} \frac{1}{m(\widetilde{\boldsymbol{a}})}$. In fact, $(s_n)_{\boldsymbol{a},\boldsymbol{a}'} = \frac{\#\{\sigma \in S_n \,;\, \forall i \leq n, \boldsymbol{a}_{\sigma(i)} = \boldsymbol{a}'_i\}}{n!} = \delta_{\widetilde{\boldsymbol{a}},\widetilde{\boldsymbol{a}'}} \frac{\prod_{a \in \mathrm{Supp}(\widetilde{\boldsymbol{a}})} (\widetilde{\boldsymbol{a}})(a)!}{n!}$.

Remark 1. The value of the matrix s_n defined above does not depend on the sequences $\boldsymbol{a}$, $\boldsymbol{a}'$ but just on their associated multisets $\widetilde{\boldsymbol{a}}$, $\widetilde{\boldsymbol{a}'}$. Hence, for any $v \in \mathbb{R}^{+|\mathcal{A}^{\otimes n}|}$, the vector $s_n\, v \in \mathbb{R}^{+|\mathcal{A}^{\otimes}|}$ can be also presented as in $\mathbb{R}^{+\mathcal{M}_n(|\mathcal{A}|)}$:

$$(s_n\, v)_\mu = \frac{1}{m(\mu)} \sum_{\boldsymbol{a} \in \mathrm{Enum}(\mu)} v_{\boldsymbol{a}}, \qquad \text{for } \mu \in \mathcal{M}_n(|\mathcal{A}|) \tag{10}$$

which is a barycentric sum because $m(\mu)$ gives exactly the cardinality of $\mathrm{Enum}(\mu)$.

This "change of base" can be explained also as follows: the image-set $s_n\,(\mathcal{A}^{\otimes n})$ of s_n is a subspace of $\mathcal{A}^{\otimes n}$ and the "canonical base" of this subspace can be given by $\mathcal{M}_n(|\mathcal{A}|)$. A vector $s_n\, v \in s_n\,(\mathcal{A}^{\otimes n}) \subseteq \mathcal{A}^{\otimes}$ can be presented then either as a family indexed by sequences (using the canonical base of $\mathcal{A}^{\otimes n}$) or as a family indexed by multisets (using the canonical base of $s_n\,(\mathcal{A}^{\otimes n})$).

Lemma 2. *The following are equivalent characterizations of a PCS $\mathcal{A}^n$ with web $|\mathcal{A}^n| = \mathcal{M}_n(|A|)$:*

1. $\mathrm{P}(\mathcal{A}^n) = \{s_n\,(\otimes_{i=1}^n u_i)\,;\; u_i \in \mathrm{P}(\mathcal{A})\}^{\perp\perp}$,
2. $\mathrm{P}(\mathcal{A}^n) = \{s_n\, u\,;\; u \in \mathrm{P}(\mathcal{A}^{\otimes n})\}$.

Proof. First, by Remark 1 notice that both sets in conditions 1 and 2 can be seen as sets of vectors over $\mathcal{M}_n(|A|)$. The case 1 is a PCS by definition, while 2 can be checked to be a PCS by means of Proposition 1. In particular, notice that 2. is Scott closed over the web $\mathcal{M}_n(|\mathcal{A}|)$, but not over $|\mathcal{A}^{\otimes n}|$.

Then, the equivalence between 1 and 2 is obtained by Lemma 1, taking $\mathcal{A} = \mathcal{A}^{\otimes n}$, $G = \{\otimes_{i=1}^n u_i\,;\; u_i \in \mathrm{P}(\mathcal{A})\}$, $\mathcal{B}$ the set described in 2 and $f = s_n$. The set described in 1 is equal to $f(G)^{\perp\perp} = f(\mathrm{P}(\mathcal{A}^{\otimes n}))^{\perp\perp}$ which turns out to be the bipolar of $\mathcal{B}$ by Lemma 1. We conclude since $\mathcal{B} = \mathcal{B}^{\perp\perp}$ as just remarked. □

Example 6. Let us illustrate the construction of $\mathcal{A}^n$ in the case where $\mathcal{A}$ is the boolean space $Bool$ (see Example 1). It is trivial to check that $Bool^0$ is isomorphic to $\mathbf{1}$ and $Bool^1$ to $Bool$. Concerning $Bool^2$, we have $\left|Bool^2\right| = \{[\mathtt{t},\mathtt{f}],[\mathtt{t},\mathtt{t}],[\mathtt{f},\mathtt{f}]\}$. Recall that Example 2 computes $\mathrm{P}\left(Bool^{\otimes 2}\right)$ as the set of sub-probability distributions over boolean pairs, so that by item 2 of Lemma 2 we have that: $\mathrm{P}\left(Bool^2\right) = \{w \in \mathbb{R}^{+^{\mathcal{M}_2(|Bool|)}} \; ; \; w_{[\mathtt{t},\mathtt{t}]} + w_{[\mathtt{f},\mathtt{f}]} + 2w_{[\mathtt{t},\mathtt{f}]} \leq 1\}$.

This characterization allows us to compute: $(\sup Bool^2)_{[\mathtt{t},\mathtt{t}]} = (\sup Bool^2)_{[\mathtt{f},\mathtt{f}]} = 1$, while $(\sup Bool^2)_{[\mathtt{t},\mathtt{f}]} = \frac{1}{2}$. Actually, using Eq. (10), one can check in general: $(\sup Bool^n)_\mu = \frac{1}{m(\mu)}$ (recall Eq. (3)). Notice that these coefficients are the inverses of the coefficients computed in Example 4.

Proposition 3. *Let $\mathcal{A}$ be a PCS and $n \in \mathbb{N}$. The object $\mathcal{A}^n$ together with the morphism $\mathrm{eq}^{\mathcal{A}^n} \in \mathbf{Pcoh}(\mathcal{A}^n, \mathcal{A}^{\otimes n})$, defined as $\mathrm{eq}^{\mathcal{A}^n}_{\mu,\boldsymbol{a}} = \delta_{\mu,\widetilde{\boldsymbol{a}}}$, is the equalizer of the $n!$ symmetries of the n-fold tensor $\mathcal{A}^{\otimes n}$.*

Moreover, these equalizers commute with the tensor product, meaning that for any $\mathcal{B}$, the morphism $\mathrm{eq}^{\mathcal{A}^n} \otimes \mathrm{id}^{\mathcal{B}}$ is the limit of the morphisms $\sigma \otimes \mathrm{id}^{\mathcal{B}}$ for $\sigma \in S_n$ (see diagram in Fig. 1b, replacing A & 1 with $\mathcal{A}$ and $A^{\leq n}$ with $\mathcal{A}^n$).

Proof. The fact that $\mathrm{eq}^{\mathcal{A}^n}$ is a morphism in $\mathbf{Pcoh}(\mathcal{A}^n, \mathcal{A}^{\otimes n})$ is immediate, because $\mathrm{eq}^{\mathcal{A}^n}$ simply maps a vector $s_n\, u$ seen as a vector over the web $\mathcal{M}_n(|\mathcal{A}|)$ to the same vector seen over the web $|\mathcal{A}^{\otimes n}|$. This latter is a vector of $\mathrm{P}\left(\mathcal{A}^{\otimes n}\right)$ (supposing $u \in \mathrm{P}\left(\mathcal{A}^{\otimes n}\right)$) because it is a barycentric sum of vectors in $\mathrm{P}\left(\mathcal{A}^{\otimes n}\right)$.

We only prove the commutation with $\otimes$, as the universal property of the equalizer is a direct consequence of the commutation, taking $\mathcal{B} = \mathbf{1}$.

Let $\mathcal{C}$ be a PCS and $f \in \mathbf{Pcoh}(\mathcal{C}, \mathcal{A}^{\otimes n} \otimes \mathcal{B})$ be a morphism such that $(\sigma \otimes \mathrm{id}^{\mathcal{B}}) \circ f = f$ for any $\sigma \in S_n$. Then, define $f^\dagger \in \mathbf{Pcoh}(\mathcal{C}, \mathcal{A}^n \otimes \mathcal{B})$ as follows: for every $c \in |\mathcal{C}|$, $b \in |\mathcal{B}|$, $\mu \in |\mathcal{A}^n|$, $f^\dagger_{c,(b,\mu)} = f_{c,(b,\boldsymbol{a})}$, where $\boldsymbol{a}$ is any enumeration of μ (no matter which one because f is invariant under the tensor symmetries). The fact that $f^\dagger$ is the unique morphism commuting the diagram of Fig. 1a is a trivial calculation. We have then to prove that it is indeed a morphism in $\mathbf{Pcoh}(\mathcal{C}, \mathcal{A}^n \otimes \mathcal{B})$, that means that, for any $v \in \mathrm{P}\left(\mathcal{C}\right)$, $f^\dagger\, v \in \mathrm{P}\left(\mathcal{A}^n \otimes \mathcal{B}\right)$.

Consider $w \in \mathrm{P}\left(\mathcal{A}^n \otimes \mathcal{B}\right)^\perp$ and let us prove that $\langle f^\dagger\, v, w\rangle \leq 1$. This will allow us to conclude $f^\dagger\, v \in \mathrm{P}\left(\mathcal{A}^n \otimes \mathcal{B}\right)$. Define $\overline{w} \in \mathbb{R}^{+^{\left|\mathcal{A}^{\otimes n} \otimes \mathcal{B}\right|}}$ as $\overline{w}_{(\boldsymbol{a},b)} = \frac{1}{m(\widetilde{\boldsymbol{a}})} w_{\widetilde{\boldsymbol{a}},b}$.

First, notice that $\overline{w} \in \mathrm{P}\left(\mathcal{A}^{\otimes n} \otimes \mathcal{B}\right)^\perp$. In fact, for any $u \in \mathrm{P}\left(\mathcal{A}^{\otimes n}\right)$, $z \in \mathrm{P}\left(\mathcal{B}\right)$:

$$\begin{aligned}\langle u \otimes z, \overline{w}\rangle &= \sum_{\boldsymbol{a},b} u_{\boldsymbol{a}} z_b \overline{w}_{(\boldsymbol{a},b)} = \sum_{\boldsymbol{a},b} u_{\boldsymbol{a}} z_b \frac{1}{m(\widetilde{\boldsymbol{a}})} w_{\widetilde{\boldsymbol{a}},b} = \sum_{\mu,b} z_b w_{\mu,b} \sum_{\boldsymbol{a} \in \mathrm{Enum}(\mu)} \frac{1}{m(\mu)} u_{\boldsymbol{a}} \\ &= \sum_{\mu,b} z_b w_{\mu,b} (s_n\, u)_\mu = \langle (s_n\, u) \otimes z, w\rangle \leq 1.\end{aligned}$$

The last inequality is due to $w \in \mathrm{P}\left(\mathcal{A}^n \otimes \mathcal{B}\right)^\perp$, $s_n\, u \in \mathrm{P}\left(\mathcal{A}^n\right)$ and Lemma 2.

Second, $\langle f^\dagger\, v, w\rangle = \langle f\, v, \overline{w}\rangle$. In fact,

$$\langle f^\dagger\, v, w\rangle = \sum_{\mu,b} (f^\dagger\, v)_{\mu,b} w_{\mu,b} = \sum_{\boldsymbol{a},b} \frac{(f\, v)_{\boldsymbol{a},b} w_{\widetilde{\boldsymbol{a}},b}}{m(\widetilde{\boldsymbol{a}})} = \sum_{\boldsymbol{a},b} (f\, v)_{\boldsymbol{a},b} \overline{w}_{\boldsymbol{a},b} = \langle f\, v, \overline{w}\rangle\ .$$

We then conclude because by hypothesis $f\, v \in \mathrm{P}\,(\mathcal{A}^{\otimes n} \otimes \mathcal{B})$. □

Given a PCS $\mathcal{A}$ and $n \in \mathbb{N}$, let us introduce the notation $\mathcal{A}^{\leq n} = (\mathcal{A}\ \&\ \mathbf{1})^n$, for the equalizer of the n-fold tensor symmetries of $\mathcal{A}\ \&\ \mathbf{1}$.

Lemma 3. *The PCS $\mathcal{A}^{\leq n}$ can be presented as follows:*

$$\left|\mathcal{A}^{\leq n}\right| = \bigcup_{k\leq n} \mathcal{M}_k(|\mathcal{A}|), \qquad \mathrm{P}\left(\mathcal{A}^{\leq n}\right) = \{\langle u_1, \ldots, u_n\rangle\ ;\ \forall i \leq n, u_i \in \mathrm{P}\,(\mathcal{A})\}^{\perp\perp}$$

where, for any $[a_1, \ldots, a_k] \in \left|\mathcal{A}^{\leq n}\right|$,

$$\langle u_1, \ldots, u_n\rangle_{[a_1,\ldots,a_k]} = \frac{1}{n!} \sum_{\sigma\in S_n} \prod_{i=1}^{k} (u_{\sigma(i)})_{a_i} \tag{11}$$

$$= \frac{(n-k)!}{n!} \sum_{f:\{1,\ldots,k\}\hookrightarrow\{1,\ldots,n\}} \prod_{i=1}^{k} (u_{f(i)})_{a_i}. \tag{12}$$

Proof. First, notice that there is a bijection between $\left|\mathcal{A}^{\leq n}\right|$ and $\mathcal{M}_n(|\mathcal{A}| \uplus \{\star\}) = |(\mathcal{A}\ \&\ \mathbf{1})^n|$ obtained just by adding the necessary number of $\star$'s to a multiset in $\left|\mathcal{A}^{\leq n}\right|$. Second, the definitions of $\mathrm{P}\left(\mathcal{A}^{\leq n}\right)$ and of $\langle u_1, \ldots, u_n\rangle$ immediately follow by remarking that the latter is a notation for $s_n\left((u_1, e_\star) \otimes \cdots \otimes (u_n, e_\star)\right)$, with s_n the endomorphism over $(\mathcal{A}\ \&\ \mathbf{1})^{\otimes n}$ defined by Eq. (10). □

Example 7. We have no simple characterization of $Bool^{\leq 2}$, but Lemma 3 helps us in computing the coefficients of its generators. For example, we have $\langle e_{\mathtt{t}}, e_{\mathtt{f}}\rangle_\mu = 1$ if $\mu = []$, otherwise $\langle e_{\mathtt{t}}, e_{\mathtt{f}}\rangle_\mu = \frac{1}{2}$. While $\langle e_{\mathtt{t}}, e_{\mathtt{t}}\rangle_\mu = 1$ for any multiset of support $\{e_{\mathtt{t}}\}$, otherwise it is 0. One can observe in general that $(\sup Bool^{\leq n})_\mu = 1$ for any multiset $\mu \in \left|Bool^{\leq n}\right|$ which is *uniform* (i.e. of which support is at most a singleton), while $(\sup Bool^{\leq n})_\mu < 1$ for μ *non-uniform*.

Henceforth, we will consider $\mathcal{A}^{\leq n}$ as presented in Lemma 3.

4.2 The Limit $!_f\mathcal{A}$

The quest for a limit $!_f\mathcal{A}$ of the family $(\mathcal{A}^{\leq n})_n$ requires the study of the relations between approximants of different degree. This is done starting from the following notions of injection and projection. Given $\mathcal{A}, \mathcal{B}$ s.t. $|\mathcal{A}| \subseteq |\mathcal{B}|$, we define the matrices *injection* $\iota_{\mathcal{A},\mathcal{B}} \in \mathbb{R}^{+|\mathcal{A}|\times|\mathcal{B}|}$ and *projection* $\rho_{\mathcal{B},\mathcal{A}} \in \mathbb{R}^{+|\mathcal{B}|\times|\mathcal{A}|}$ as follows:

$$(\iota_{\mathcal{A},\mathcal{B}})_{a,b} = (\rho_{\mathcal{B},\mathcal{A}})_{b,a} = \delta_{a,b}. \tag{13}$$

The injection $\iota_{\mathcal{A},\mathcal{B}}$ maps a vector $u \in \mathbb{R}^{+|\mathcal{A}|}$ to the vector $(u, \mathbf{0}) \in \mathbb{R}^{+|\mathcal{B}|}$ associating the directions in $|\mathcal{B}| \setminus |\mathcal{A}|$ with zero. The projection $\rho_{\mathcal{B},\mathcal{A}}$ maps a vector $(u, v) \in \mathbb{R}^{+|\mathcal{B}|}$ to its restriction u to the directions within $|\mathcal{A}|$. In order to have these matrices as morphisms in **Pcoh**, we have to prove that $(u, \mathbf{0}) \in \mathrm{P}(\mathcal{B})$ whenever $u \in \mathrm{P}(\mathcal{A})$ (resp. $u \in \mathrm{P}(\mathcal{A})$ whenever $(u, v) \in \mathrm{P}(\mathcal{B})$).

As already remarked in Definition 2, the projection of $\mathcal{A}^{\leq n+1}$ into $\mathcal{A}^{\leq n}$ is actually obtained by applying the universal property to the morphism $(\mathtt{id} \otimes \mathrm{pr}_r) \circ \mathtt{eq}$. So we get the following immediate lemma.

Lemma 4. *For any $m \geq n$ and $\mathcal{A}$, we have: $\rho_{\mathcal{A}^{\leq m},\mathcal{A}^{\leq n}} \in \mathbf{Pcoh}(\mathcal{A}^{\leq m}, \mathcal{A}^{\leq n})$ and $\iota_{\mathcal{A}^{\leq n\perp},\mathcal{A}^{\leq m\perp}} \in \mathbf{Pcoh}(\mathcal{A}^{\leq n\perp}, \mathcal{A}^{\leq m\perp})$.*

The interesting point is that the dual version of Lemma 4 does not hold: in general, the injection of $\mathcal{A}^{\leq n}$ into $\mathcal{A}^{\leq n+1}$ (resp. projection of $(\mathcal{A}^{\leq n+1})^\perp$ into $(\mathcal{A}^{\leq n})^\perp$) is not a morphism of **Pcoh**.

Example 8. In Example 7, we discussed $\langle e_\mathtt{t}, e_\mathtt{f}\rangle \in \mathrm{P}\left(Bool^{\leq 2}\right)$. Let us prove now that $\iota_{Bool^{\leq 2},Bool^{\leq 3}} \langle e_\mathtt{t}, e_\mathtt{f}\rangle \notin \mathrm{P}\left(Bool^{\leq 3}\right)$. In fact, $(\iota_{Bool^{\leq 2},Bool^{\leq 3}} \langle e_\mathtt{t}, e_\mathtt{f}\rangle)_{[\mathtt{t},\mathtt{f}]} = \langle e_\mathtt{t}, e_\mathtt{f}\rangle_{[\mathtt{t},\mathtt{f}]} = \frac{1}{2}$, while we can prove $(\sup \mathrm{P}\left(Bool^{\leq 3}\right))_{[\mathtt{t},\mathtt{f}]} = \frac{1}{3}$. The latter claim is because $\mathrm{P}\left(Bool^{\leq 3}\right) = \{\langle e_\mathtt{t}, e_\mathtt{t}, e_\mathtt{t}\rangle, \langle e_\mathtt{t}, e_\mathtt{t}, e_\mathtt{f}\rangle, \langle e_\mathtt{t}, e_\mathtt{f}, e_\mathtt{f}\rangle, \langle e_\mathtt{f}, e_\mathtt{f}, e_\mathtt{f}\rangle\}^{\perp\perp}$ and the maximal value of these generators on $[\mathtt{t}, \mathtt{f}]$ is $\frac{1}{3}$.

One can however add a correction factor in order to embed $\mathcal{A}^{\leq n}$ into $\mathcal{A}^{\leq N}$ for any $N \geq n$, as follows. Let:

$$(\iota^{cor}_{\mathcal{A},n,N})_{\mu,\nu} = \begin{cases} \frac{(N-k)!q^k n!}{N!(n-k)!} & \text{if } \mu = \nu \text{ and } \#\mu = k \leq n \\ & \text{and } q = \lfloor \frac{N}{n} \rfloor \text{ and } r = N \bmod n; \\ 0 & \text{otherwise.} \end{cases} \tag{14}$$

Lemma 5. *The matrix $\iota^{cor}_{\mathcal{A},n,N}$ is a morphism in $\mathbf{Pcoh}(\mathcal{A}^{\leq n}, \mathcal{A}^{\leq N})$ mapping, in particular, $\langle u_1, \ldots, u_n\rangle \in \mathrm{P}\left(\mathcal{A}^{\leq n}\right)$ to $\langle u_1^q, \ldots, u_n^q, \mathbf{0}^r\rangle \in \mathrm{P}\left(\mathcal{A}^{\leq N}\right)$, where q is the quotient $\lfloor \frac{N}{n} \rfloor$ and r the remainder $N \bmod n$ of the euclidean division $\frac{N}{n}$. Moreover, u_i^q is a notation for $\underbrace{u_i, \ldots, u_i}_{q \text{ times}}$ (and similarly for $\mathbf{0}^r$).*

Proof. One has just to prove the last part of the statement, the rest follows by Lemma 1 because the vectors of the form $\langle u_1, \ldots, u_n\rangle$ yield a base for $\mathrm{P}\left(\mathcal{A}^{\leq n}\right)$ (Lemma 3). We have, for any multiset $\mu = [a_1, \ldots, a_k]$,

$$\begin{aligned}(\iota^{cor}_{\mathcal{A},n,N} \langle u_1, \ldots, u_n\rangle)_\mu &= \left(\frac{(N-k)!q^k n!}{N!(n-k)!}\right) \frac{(n-k)!}{n!} \sum_{f:\{1,\ldots,k\}\hookrightarrow\{1,\ldots,n\}} \prod_{i=1}^{k} (u_{f(i)})_{a_i} \\ &= \frac{(N-k)!}{N!} q^k \sum_{f:\{1,\ldots,k\}\hookrightarrow\{1,\ldots,n\}} \prod_{i=1}^{k} (u_{f(i)})_{a_i}\end{aligned}$$

$$= \frac{(N-k)!}{N!} \sum_{f:\{1,\dots,k\}\hookrightarrow\{1,\dots,nq\}} \prod_{i=1}^{k} (u_{(\lfloor f(i)/q \rfloor)+1})_{a_i}$$
$$= \langle u_1^q, \dots, u_n^q, \mathbf{0}^r \rangle_\mu.$$

where we use $\hookrightarrow$ to denote injective functions and where from lines 2 to 3, we use the count q^k to enlarge the codomain of the injections f indexing the sum.□

Notice that, with $N = nq + r$:

$$\lim_{N\to\infty} \frac{(N-k)!q^k n!}{N!(n-k)!} = \frac{n!}{n^k(n-k)!}, \tag{15}$$

$$(\lim_{N\to\infty} \iota^{cor}_{\mathcal{A},n,N} \langle u_1, \dots, u_n \rangle)_\mu = \frac{1}{n^k} \sum_{f:\{1,\dots,k\}\hookrightarrow\{1,\dots,n\}} \prod_{i=1}^{k} (u_{f(i)})_{a_i}. \tag{16}$$

This allows us to introduce the following definition and key proposition:

Definition 3. *Given $\mathcal{A}$, we define $!_f\mathcal{A}$ as:*

$$|!_f\mathcal{A}| = \mathcal{M}_f(|\mathcal{A}|), \quad \mathrm{P}(!_f\mathcal{A}) = \{\langle\!\langle u_1, \dots, u_n \rangle\!\rangle \; ; \; n \in \mathbb{N}, \forall i \leq n, u_i \in \mathrm{P}(\mathcal{A})\}^{\perp\perp},$$

where: $\langle\!\langle u_1, \dots, u_n \rangle\!\rangle_{[a_1,\dots,a_k]} = \frac{1}{n^k} \sum_{f:\{1,\dots,k\}\hookrightarrow\{1,\dots,n\}} \prod_{i=1}^{k} (u_{f(i)})_{a_i}$.

Notice that whenever $k > n$, $\langle\!\langle u_1, \dots, u_n \rangle\!\rangle_{[a_1,\dots,a_k]} = 0$.

Proposition 4. *Let $\mathcal{A}$ be a PCS. The object $!_f\mathcal{A}$ together with the family of morphisms $\rho_{!_f\mathcal{A},\mathcal{A}^{\leq n}} \in \mathbf{Pcoh}(!_f\mathcal{A}, \mathcal{A}^{\leq n})$ for $n \in \mathbb{N}$, constitute the limit of the chain $\mathbf{1} \xleftarrow{\rho} \mathcal{A}^{\leq 1} \xleftarrow{\rho} \mathcal{A}^{\leq 2} \xleftarrow{\rho} \dots$.*

Moreover, this limit commutes with the tensor product (Fig. 2).

Proof. First, we prove that $\rho_{!_f\mathcal{A},\mathcal{A}^{\leq m}}$ is a correct morphism mapping $\mathrm{P}(!_f\mathcal{A})$ into $\mathrm{P}(\mathcal{A}^{\leq m})$, for any m. By Lemma 1 it is enough to check that any $\langle\!\langle u_1, \dots, u_n \rangle\!\rangle$ is mapped to the pcs $\mathrm{P}(\mathcal{A}^{\leq m})$ by $\rho_{!_f\mathcal{A},\mathcal{A}^{\leq m}}$. As $\mathrm{P}(\mathcal{A}^{\leq m}) = \mathrm{P}(\mathcal{A}^{\leq m})^{\perp\perp}$, it is equivalent to show that for any $n \in \mathbb{N}$, $u_i \in \mathrm{P}(\mathcal{A})$, and $w \in \mathrm{P}(\mathcal{A}^{\leq m})^{\perp}$ we have:

$$\langle \rho_{!_f\mathcal{A},\mathcal{A}^{\leq m}} \langle\!\langle u_1, \dots, u_n \rangle\!\rangle, w \rangle \leq 1 \tag{17}$$

Lemma 4 and 5 give us, $\forall N \geq m, n$, resp.: $\iota_{\mathcal{A}^{\leq m},\mathcal{A}^{\leq N}} w \in \mathrm{P}(\mathcal{A}^{\leq N})^{\perp}$ and $\iota^{cor}_{\mathcal{A},n,N} \langle u_1, \dots, u_n \rangle \in \mathrm{P}(\mathcal{A}^{\leq N})$. Thus the inner product between the two vectors is bounded by 1. Consider then the limit of this product for $N \to \infty$:

$$\begin{aligned}
1 \geq & \lim_{N\to\infty} \langle \iota^{cor}_{\mathcal{A},n,N} \langle u_1, \dots, u_n \rangle, \iota_{\mathcal{A}^{\leq m},\mathcal{A}^{\leq N}} w \rangle \\
= & \langle \langle\!\langle u_1, \dots, u_n \rangle\!\rangle, \iota_{\mathcal{A}^{\leq m},!_f\mathcal{A}} w \rangle && \text{(Eq. (16) and Definition 3)} \\
= & \langle \rho_{!_f\mathcal{A},\mathcal{A}^{\leq m}} \langle\!\langle u_1, \dots, u_n \rangle\!\rangle, (\rho_{!_f\mathcal{A},\mathcal{A}^{\leq m}} \circ \iota_{\mathcal{A}^{\leq m},!_f\mathcal{A}}) w \rangle \\
= & \langle \rho_{!_f\mathcal{A},\mathcal{A}^{\leq m}} \langle\!\langle u_1, \dots, u_n \rangle\!\rangle, w \rangle
\end{aligned}$$

Line 2 gives line 3 using of the definition of ι and ρ in (13).

Now we prove that $!_f\mathcal{A}$ together with its projections $\rho_{!_f\mathcal{A},\mathcal{A}^{\leq m}}$ is indeed a limit cone. As for Proposition 3, we prove straight the commutation with the $\otimes$, as the first part of the statement is a consequence of this latter, taking $\mathcal{B} = \mathbf{1}$.

Take a PCS $\mathcal{C}$ and an $\mathbb{N}$-indexed family of morphisms $f_n \in \mathbf{Pcoh}(\mathcal{C}, \mathcal{A}^{\leq n} \otimes \mathcal{B})$ commuting with the chain $\mathcal{B} \xleftarrow{\rho} \mathcal{A}^{\leq 1} \otimes \mathcal{B} \xleftarrow{\rho} \mathcal{A}^{\leq 2} \otimes \mathcal{B} \xleftarrow{\rho} \dots$. We should define a unique $f^\dagger$ s.t. Fig. 2 commutes. The matrix $f^\dagger$ is defined as: $f^\dagger_{c,(\mu,b)} = (f_{\#\mu})_{c,(\mu,b)}$. The fact that $f^\dagger$ is the unique one such that Fig. 2 commutes is an easy calculation. We should then prove that it is a morphism in $\mathbf{Pcoh}(\mathcal{C}, !_f\mathcal{A}\otimes\mathcal{B})$, i.e. for every $v \in \mathrm{P}(\mathcal{C})$, $f^\dagger\, v \in \mathrm{P}(!_f\mathcal{A} \otimes \mathcal{B})$. Since $\mathrm{P}(!_f\mathcal{A} \otimes \mathcal{B}) = \mathrm{P}(!_f\mathcal{A} \otimes \mathcal{B})^{\perp\perp}$, it is equivalent to prove that: $\forall w \in \mathrm{P}(!_f\mathcal{A} \otimes \mathcal{B})^\perp$, $\langle f^\dagger\, v, w\rangle \leq 1$.

For any n, define $w \downarrow_n \in \mathbb{R}^{+\left|\mathcal{A}^{\leq n} \otimes \mathcal{B}\right|}$ as:

$$(w \downarrow_n)_{(\mu,b)} = \begin{cases} \frac{n!}{n^k(n-k)!} w_{(\mu,b)} & \text{if } \#\mu = k \leq n, \\ 0 & \text{otherwise.} \end{cases}$$

Notice that, for any $\langle u_1, \dots, u_n\rangle \in \mathrm{P}\left(\mathcal{A}^{\leq n}\right)$ and $z \in \mathrm{P}(\mathcal{B})$, we have the inequality: $\langle\langle u_1, \dots, u_n\rangle \otimes z, w \downarrow_n\rangle = \langle\langle\!\langle u_1, \dots, u_n\rangle\!\rangle \otimes z, w\rangle \leq 1$. We conclude that $w \downarrow_n \in \mathrm{P}\left(\mathcal{A}^{\leq n} \otimes \mathcal{B}\right)^\perp$, for any n. Then we have:

$$\begin{aligned} 1 \geq \lim_{n\to\infty} \langle f_n\, v, w \downarrow_n\rangle &= \lim_{n\to\infty} \langle (\rho_{!_f\mathcal{A},\mathcal{A}^{\leq n}} \circ f^\dagger)\, v, w \downarrow_n\rangle && (\text{def } f^\dagger) \\ &= \langle f^\dagger\, v, w\rangle\ . && (\text{Eq. (16)}) \end{aligned}$$

□

Propositions 2, 3 and 4 give the last corollary.

Corollary 1. *For any PCS $\mathcal{A}$, the PCS $!_f\mathcal{A}$ yields the free commutative comonoid generated by $\mathcal{A}$.*

4.3 The Free and Entire Exponential Modalities Are the Same

How do the approximants $\mathcal{A}^{\leq n}$ of $!_f\mathcal{A}$ relate with $!_e\mathcal{A}$? Let us consider $\mathcal{A} = \mathit{Bool}$, and compare the maximal coefficients of these spaces on $[\mathtt{t}, \mathtt{f}]$. It is easy to check that $(\sup !_e\mathit{Bool})_{[\mathtt{t},\mathtt{f}]} = (\frac{e_\mathtt{t}+e_\mathtt{f}}{2})^!_{[\mathtt{t},\mathtt{f}]} = \frac{1}{4}$. While $(\sup \mathit{Bool}^{\leq n})_{[\mathtt{t},\mathtt{f}]} = (\langle e_\mathtt{t}^{\lfloor \frac{n}{2} \rfloor}, e_\mathtt{f}^{\lceil \frac{n}{2} \rceil}\rangle_{[\mathtt{t},\mathtt{f}]} = (\langle e_\mathtt{t}^{\lceil \frac{n}{2} \rceil}, e_\mathtt{f}^{\lfloor \frac{n}{2} \rfloor}\rangle_{[\mathtt{t},\mathtt{f}]}$, whose values are, for $n = 2, 3, 4, \dots$ (using Eq. (12)): $\frac{1}{2}, \frac{1}{3}, \frac{1}{3}, \frac{3}{10}, \frac{3}{10}, \frac{2}{7}, \dots, \frac{1}{n(n-1)} \lfloor \frac{n}{2} \rfloor \lceil \frac{n}{2} \rceil \dots$ converging to $\frac{1}{4}$. This remark can be generalized, showing that the approximants $\mathcal{A}^{\leq n}$ are actually approaching to $!_e\mathcal{A}$ from above, giving that their limit is equal to $!_e\mathcal{A}$.

Proposition 5. *For any PCS $\mathcal{A}$, we have $!_f A = !_e A$.*

Proof. The two spaces have the same web, we prove that $\mathrm{P}(!_f A) = \mathrm{P}(!_e A)$.

Concerning $\mathrm{P}(!_f A) \subseteq \mathrm{P}(!_e A)$. Take any $\langle\!\langle u_1, \dots, u_n\rangle\!\rangle \in \mathrm{P}(!_f A)$, we have $\langle\!\langle u_1, \dots, u_n\rangle\!\rangle \in \mathrm{P}(!_e A)$, because $\langle\!\langle u_1, \dots, u_n\rangle\!\rangle \leq (\frac{1}{n}\sum_i u_i)^! \in \mathrm{P}(!_e A)$.

Conversely, let u^n denotes $u, \dots, u$ repeated n times in

$$\langle\!\langle u^n \rangle\!\rangle_{[a_1,\dots,a_k]} = \frac{n!}{n^k(n-k)!} \prod_{i=1}^{k} u_{a_i} = \frac{n!}{n^k(n-k)!} u^!_{[a_1,\dots,a_k]}.$$

As for $k < n$, $\frac{n!}{n^k(n-k)!}$ is an increasing sequence converging to 1, we get that $\forall u \in \mathrm{P}(A)$, $\sup_n \langle\!\langle u^n \rangle\!\rangle = u^!$. Now, $u^! \in \mathrm{P}(!_f \mathcal{A})$ since it is Scott-closed. Since $u^!$ for $u \in \mathrm{P}(\mathcal{A})$ are generating $\mathrm{P}(!_e \mathcal{A})$, we conclude that $\mathrm{P}(!_e A) \subseteq \mathrm{P}(!_f A)$. □

Acknowledgments. We thank Sam Staton, Hugh Steele, Lionel Vaux and the anonymous reviewers for useful comments and discussions. This work has been partly funded by the French project ANR-14-CE25-0005 *Elica* and by the French-Chinese project ANR-11-IS02-0002 and NSFC 61161130530 *Locali*.

References

1. Breuvart, F., Pagani, M.: Modelling coeffects in the relational semantics of linear logic. In: Kreutzer, S. (ed.) Proceedings of the 24th EACSL Annual Conference on Computer Science Logic, CSL15, Berlin, Germany. LIPICS (2015)
2. Carraro, A., Ehrhard, T., Salibra, A.: Exponentials with infinite multiplicities. In: Dawar, A., Veith, H. (eds.) CSL 2010. LNCS, vol. 6247, pp. 170–184. Springer, Heidelberg (2010). doi:10.1007/978-3-642-15205-4_16
3. Danos, V., Ehrhard, T.: Probabilistic coherence spaces as a model of higher-order probabilistic computation. Inf. Comput. **209**(6), 966–991 (2011)
4. Ehrhard, T., Pagani, M., Tasson, C.: Probabilistic coherence spaces are fully abstract for probabilistic PCF. In: Sewell, P. (ed.) The 41th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL14, San Diego, USA. ACM (2014)
5. Ehrhard, T., Tasson, C.: Probabilistic call by push value (2016). preprint http://arxiv.org/abs/1607.04690
6. Girard, J.-Y.: Linear logic. Theor. Comput. Sci. **50**, 1–102 (1987)
7. Girard, J.-Y.: Linear logic: its syntax and semantics. In: Girard, J.-Y., Lafont, Y., Regnier, L. (eds.) Advances in Linear Logic, London Math. Soc. Lect. Notes Ser, vol. 222, pp. 1–42. (1995)
8. Girard, J.-Y.: Between logic and quantic: a tract. In: Ehrhard, T., Girard, J.-Y., Ruet, P., Scott, P. (eds.) Linear Logic in Computer Science, London Math. Soc. Lect. Notes Ser., vol. 316. CUP (2004)
9. Lafont, Y.: Logiques, catégories et machines. Ph.D. thesis, Université Paris 7 (1988)
10. Melliès, P.-A.: Categorical semantics of linear logic. Panoramas et Synthèses, 27 (2009)
11. Melliès, P.-A., Tabareau, N., Tasson, C.: An explicit formula for the free exponential modality of linear logic. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 247–260. Springer, Heidelberg (2009). doi:10.1007/978-3-642-02930-1_21
12. van de Wiele, J.: Manuscript (1987)

From Qualitative to Quantitative Semantics
By Change of Base

James Laird(✉)

Department of Computer Science, University of Bath, Bath, UK
jiml@cs.bath.ac.uk

Abstract. We give a general description of the transition from qualitative models of programming languages to quantitative ones, as a change of base for enriched categories. This is induced by a monoidal functor from the category of coherence spaces to the category of modules over a complete semiring $\mathcal{R}$. Using the properties of this functor, we characterise the requirements for the change of base to preserve the structure of a Lafont category (model of linear type theory with free exponential), and thus to give an adequate semantics of erratic PCF with scalar weights from $\mathcal{R}$. Moreover, this model comes with a meaning-preserving functor from the original, qualitative one, which we may use to interpret side-effects such as state. As an example, we show that the game semantics of Idealized Algol bears a natural enrichment over the category of coherence spaces, and thus gives rise by change of base to a $\mathcal{R}$-weighted model, which is fully abstract. We relate this to existing categories of probabilistic games and slot games.

1 Introduction

Game semantics have been used to successfully describe intensional models of a wide variety of programming language features. With some notable (generally ad-hoc) exceptions, these models are qualitative rather than quantitative in character, possessing an order-theoretic structure which may be characterized as a categorical *enrichment* over certain categories of domain (such as dI-domains, qualitative domains and prime algebraic lattices). Our aim is to show that this enriched category theory perspective may be used to systematically construct quantitative models (and describe existing ones), using the notion of *change of base* to vary the enrichment of the model, independently of its intensional structure. Specifically, we describe a monoidal functor from a category of coherence spaces to the category of $\mathcal{R}$-weighted relations, where $\mathcal{R}$ is a complete semiring. The change of base induced by this functor allows a semantic translation from a qualitative model, enriched over coherence spaces, to a quantitative one in which program denotations are weighted with values in $\mathcal{R}$ corresponding to (e.g.) measures of probability, security, resource usage, etcetera. We illustrate this by example, showing that the well-known games model of Idealized Algol [1] bears

J. Laird—Research Supported by UK EPSRC Grant EP/K037633/1.

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 36–52, 2017.
DOI: 10.1007/978-3-662-54458-7_3

a natural enrichment over coherence spaces, and describing the fully abstract semantics of $\mathcal{R}$-weighted Idealized Algol which we obtain from it by change of base.

Related Work. The monoidal categories we use for enrichment are based on existing, extensonal models of computation, and logic — on the qualitative side, coherence spaces or qualitative domains [12]. On the quantitative side, we use the category of weighted relations over a complete semiring $\mathcal{R}$ (equivalently, free $\mathcal{R}$-modules and linear functions). This was introduced as a model of linear logic by Lamarche [21] and its computational properties studied in [20] via a semantics of $\mathcal{R}$-weighted PCF.

A more general categorical characterization of these quantitative models, abstracting their key properties, was given in [18]: any model of intuitionistic linear logic with a *free exponential* and (countably) infinite biproducts yields an adequate model of PCF weighted over its *internal semiring*. To arrive at such a category using the change of base, we need to show that it preserves the free exponential. However, this is not an enriched functor: instead, we give conditions under which the construction of the cofree commutative comonoid given in [22], as a limit of symmetric tensor powers, is preserved.

Change of base thus provides a simple way to identify further examples of this categorical model, with richer internal structure than sets and weighted relations. This will allow more language features such as side effects to be captured, and also provide a way to attack the *full abstraction* problem for these models (the weighted relational models for PCF are shown not to be fully abstract in [20]). As an illustrative example we study the games model introduced in [1]. We show that a strictly linear (rather than affine) version of this category of games bears a natural enrichment over coherence spaces — as foreshadowed by in [5] by a projection into a category of (ordered) coherence spaces. Previous quantitative models based on this category of games include Danos and Harmer's *probabilistic games* [8], in which strategies are defined by attaching probabilities to positions of the game, and Ghica's *slot games* [11], which attach resource weightings to positions in a rather different way — by introducing a class of moves which are persistent when other moves are hidden during composition, allowing the cost of computation to be made explicit. We show that both may be viewed as examples obtained by our change of base construction, and that the corresponding programming languages (Algol weighted with probabilities, and resource costs) may be subsumed into a version of Idealized Algol with weights from a complete semiring, for which we describe a denotational semantics. Full abstraction for this model follows from the result in [1] with very little effort.

1.1 Enriched Categories and Change of Base

Recall that if $\mathcal{V}$ is a monoidal category, then a $\mathcal{V}$-category $\mathcal{C}$ is given by a set of its objects, a $\mathcal{V}$-object $\mathcal{C}(A, B)$ for each pair of $\mathcal{C}$-objects A and B, and $\mathcal{V}$-morphisms $\mathsf{comp}_{A,B,C} : \mathcal{V}(A, B) \otimes \mathcal{V}(B, C) \to \mathcal{V}(A, C)$ and $\mathsf{id}_A : I \to \mathcal{C}(A, A)$ for each A, B, C, satisfying the expected associativity and identity diagrams in $\mathcal{V}$.

Intensional semantics such as games models may be represented as $\mathcal{V}$-categories, where $\mathcal{V}$ captures some relations or operations on morphisms such as partial orders or algebraic structures. This gives an extensional characterization of the model which can be studied independently of the intensional aspect. An example is the notion of *change of base* [7], which uses a monoidal functor from $\mathcal{V}$ to $\mathcal{W}$ to transform any model in a $\mathcal{V}$-category to a model in a $\mathcal{W}$-category satisfying the same equational theory.

More precisely, given monoidal categories $\mathcal{V}$ and $\mathcal{W}$ any monoidal functor $(F, m) : \mathcal{V} \to \mathcal{W}$ induces a change of base which takes each $\mathcal{V}$-category $\mathcal{C}$ to the $\mathcal{W}$-category $F_*(\mathcal{C})$ over the same objects, with $F_*(\mathcal{C})(A, B) = F(\mathcal{C}(A, B))$, and composition and identity morphisms $m_{\mathcal{C}(A,B),\mathcal{C}(B,C)}; F(\mathsf{comp}_{A,B,C})$ and $m_I; F(\mathsf{id}_A)$. A simple example is the change of base induced by the monoidal functor $\mathcal{V}(I, \_) : \mathcal{V} \to \mathbf{Set}$, which sends each $\mathcal{V}$-category $\mathcal{C}$ to its *underlying category* $\mathcal{C}_0$. Note change of base induced by the monoidal functor F comes with an identity-on objects, F-on-morphisms functor $F_0 : \mathcal{C}_0 \to F_*(\mathcal{C})_0$.

Change of base preserves enriched functors and natural transformations, giving a 2-functor F_* from the category of $\mathcal{V}$-categories to the category of $\mathcal{W}$-categories. Thus, in particular, it preserves symmetric monoidal structure, and symmetric monoidal closure (the existence of a natural $\mathcal{V}$-isomorphism $\mathcal{C}(A \otimes B, C) \cong \mathcal{C}(A, B \multimap C)$. A second example: if $\mathcal{V}$ is symmetric monoidal closed, and therefore enriched over itself, then any monoidal functor $F : \mathcal{V} \to \mathcal{W}$ induces a change of base to a $\mathcal{W}$-enriched symmetric monoidal closed category.

2 Coherence Spaces and Weighted Relations

We will describe a monoidal functor from the category of coherence spaces and stable, continuous linear functions to the category of sets and weighted relations (a.k.a. free $\mathcal{R}$-modules). Examples of categories of intensional models which may be enriched over coherence spaces or qualitative domains are common. For instance any symmetric monoidal closed category with a monoidal functor into the category of coherence spaces gives a coherence space enriched category as noted above — examples include categories of hypercoherences [9], event structures [24], concrete data structures [4] and games [5]. (However, the enriched category of games that we describe does not arise in this way.)

A coherence space [12] D is a pair $(|D|, \frown_D)$ where $|D|$ is a set of atoms (the web), and $\frown_D \subseteq |D| \times |D|$ is a symmetric and reflexive relation (coherence). A clique X of D is a set of its atoms which is pairwise coherent: $d, d' \in X \implies d \frown_D d'$.

The symmetric monoidal (closed) category **CSpace** has coherence spaces as objects: morphisms from D to E are cliques of the coherence space $D \multimap E$, where $|D \multimap E| = |D| \times |E|$ and $(d, e) \frown_{D \multimap E} (d', e')$ if $d \frown_D d'$ implies $e \frown_E e' \wedge (d \neq d' \vee d = d')$.

In other words, morphisms are certain relations between webs, and are composed accordingly — if $f : C \to D$ and $g : D \to E$ then $f; g = \{(c, e) \in |C| \times |E| \mid \exists d \in D.(c, d) \in f \wedge (d, e) \in g\}$. Evidently, the identity relation is a

clique. The tensor product of coherence spaces is the cartesian product of their webs and coherence relations — i.e. $|D \otimes E| = |D| \times |E|$, with $(d, e) \frown_{D \otimes E} (d', e')$ if and only if $d \frown_D d'$ and $e \frown_E e'$. The tensor unit is the singleton coherence space $\{*\}$.

The cliques of a coherence space E form an atomistic (Scott) domain, and the morphisms of **CSpace** correspond to linear, continuous and stable morphisms between the corresponding domains (i.e. preserving suprema of all directed and bounded sets, and infima of finite bounded sets).

Complete Semirings and Weighted Relations. We now recall categories of weighted relations, which will be used to characterise the structure of quantitative models. They are based on monoids with an infinitary "sum" operation.

Definition 1. *A* complete monoid *is a pair (S, Σ) of a set S with a sum operation Σ on indexed families of elements of S, satisfying the axioms:*

Partition Associativity. *For any partitioning of the set I into $\{I_j \mid j \in J\}$, $\Sigma_{i \in I} a_i = \Sigma_{j \in J} \Sigma_{i \in I_j} a_i$.*
Unary Sum. $\Sigma_{i \in \{j\}} a_i = a_j$.

We write 0 for the sum of the empty family, which is a neutral element for the sum by the above axioms.

Definition 2. *A (commutative)* complete semiring $\mathcal{R}$ *is a tuple $(|\mathcal{R}|, \Sigma, \cdot, 1)$ such that $(|\mathcal{R}|, \Sigma)$ is a complete monoid and $(|\mathcal{R}|, \cdot, 1)$ is a commutative monoid which distributes over Σ — i.e. $\Sigma_{i \in I}(a \cdot b_i) = a \cdot \Sigma_{i \in I} b_i$.*

A $\mathcal{R}$-module is a monoidal action ("scalar multiplication") of $(|\mathcal{R}|, \cdot, 1)$ on a complete monoid (S, Σ), which is distributive on both sides — i.e. $(\Sigma_{i \in I} a_i).v = \Sigma_{i \in I} a_i.v$ and $a.\Sigma_{i \in I} v_i = \Sigma_{i \in I} a.v_i$.

For any complete semiring the forgetful functor from the category of $\mathcal{R}$-modules and their homomorphisms into the category of sets has a left adjoint, which sends a set X to the "free semimodule" $\mathcal{R}^X$, which is the set of functions from X into $\mathcal{R}$, with the sum and scalar product defined pointwise. Resolving this adjunction gives a commutative monad $\mathcal{R}^-$ on the category of sets and thus a co-Kleisli category $\mathbf{Set}_{\mathcal{R}}$ with symmetric monoidal structure (given by the product of sets). Morphisms from X to Y in this category correspond both to $\mathcal{R}$-module homomorphisms from $\mathcal{R}^X$ to $\mathcal{R}^Y$, and also to $\mathcal{R}$-*weighted relations*, with which we will henceforth identify them: maps from $X \times Y$ into $\mathcal{R}$, composed by setting $(f; g)(x, z) = \Sigma_{y \in Y} f(x, y) \cdot g(y, z)$. The symmetric monoidal action on weighted relations is $(f \otimes g)((u, v), (x, y)) = f(u, x) \cdot g(v, y)$.

Relations weighted with *continuous* semirings are discussed in [20], with examples including any complete lattice, the natural or positive real numbers completed with a greatest element ∞, and the so-called *exotic* semirings. Examples of complete but not continuous semiring weights are considered in [18].

2.1 From Cliques to Weighted Relations

Note that composition of morphisms $f : C \to D$ and $g : D \to F$ in **CSpace** has the following property, derived from stability.

Lemma 1. *$(c, e) \in f; g$ if and only if there exists a* unique *$d \in D$ such that $(c, d) \in f$ and $(d, e) \in g$.*

Proof. Existence holds by definition. For uniqueness, suppose $(c, d), (c, d') \in f$ and $(d, e), (d', e) \in g$. Then $d \subset\!\!\!\supset_D d'$ and hence $d = d'$.

We define a functor $\Phi^{\mathcal{R}} : \mathbf{CSpace} \to \mathbf{Set}_{\mathcal{R}}$ which sends each object to its underlying set, and each morphism from D to E to its characteristic function — i.e. $\Phi^{\mathcal{R}}(f)(c, d) = 1$, if $(c, d) \in f$; $\Phi^{\mathcal{R}}(f)(c, d) = 0$, otherwise.

By definition, $\Phi^{\mathcal{R}}(\mathsf{id}_D)$ is the identity on $|D|$ in $\mathbf{Set}_{\mathcal{R}}$. Thus functoriality follows from Lemma 1.

Lemma 2. $\Phi^{\mathcal{R}}(f); \Phi^{\mathcal{R}}(g) = \Phi^{\mathcal{R}}(f) : \Phi^{\mathcal{R}}(g)$.

Proof. Suppose $\Phi^{\mathcal{R}}(f; g)(c, e) = 1$ — i.e. $(c, e) \in f; g$. By Lemma 1 there exists a unique $d \in D$ such that $(c, d) \in f$ and $(d, e) \in g$. Thus $\Phi^{\mathcal{R}}(f)(c, d').\Phi^{\mathcal{R}}(f)(c, d') = 1$ if $d = d'$ and $\Phi^{\mathcal{R}}(f)(c, d').\Phi^{\mathcal{R}}(f)(c, d') = 0$ otherwise. Hence $\Phi^{\mathcal{R}}(f); \Phi^{\mathcal{R}}(g)(c, e) = \Sigma_{d \in D} \Phi^{\mathcal{R}}(f)(c, d).\Phi^{\mathcal{R}}(g)(d, e) = 1$.

Otherwise $\Phi^{\mathcal{R}}(f; g)(c, e) = 0$ — i.e. $(c, e) \notin f; g$, so that for all $d \in D$, either $(c, d) \notin f$ or $(d, e) \notin g$ and so $\Phi^{\mathcal{R}}(f)(c, d).\Phi^{\mathcal{R}}(f)(c, d) = 0$. Then $\Phi^{\mathcal{R}}(f); \Phi^{\mathcal{R}}(g)(c, e) = \Sigma_{d \in D} \Phi^{\mathcal{R}}(f)(c, d).\Phi^{\mathcal{R}}(g)(d, e)$.

Evidently, Φ^R is *strict monoidal* — $\Phi^R(I) = I$ and $\Phi^{\mathcal{R}}(D \otimes E) = \Phi^{\mathcal{R}}(D) \otimes \Phi^{\mathcal{R}}(E)$, and faithful. Thus, by change of base, for each ordered complete semiring $\mathcal{R}$ we have a 2-functor Φ^R_* from the category of (symmetric monoidal closed) **CSpace**-categories to the category of (symmetric monoidal closed) $\mathbf{Set}_{\mathcal{R}}$-categories, with a faithful functor $\Phi^R_0 : \mathcal{C}_0 \to \Phi^R_*(\mathcal{C})_0$ for each **CSpace**-category $\mathcal{C}$.

Remark 1. If $\mathcal{R}$ is *idempotent* ($a_i = a$ for all $i \in I$ (non-empty) implies $\Sigma_{i \in I} a_i$) then functoriality no longer depends on Lemma 1 and thus the stability of morphisms. Hence we may define a monoidal functor from the category of sets and relations to the category of $\mathcal{R}$-weighted relations which sends each relation to its characteristic function, yielding a change of base from **Rel**-enriched to $\mathbf{Set}_{\mathcal{R}}$ enriched categories whenever $\mathcal{R}$ is idempotent.

3 An Example: Games and History-Sensitive Strategies

We illustrate by describing an example of a family of quantitative games models obtained by change of basis applied to a symmetric monoidal category of games and "knowing" strategies enriched over coherence spaces. Its underlying category is essentially the games model of *Idealized Algol* (IA) introduced by Abramsky and McCusker in [1] and obtained by relaxing the *innocence* constraint on strategies in the Hyland-Ong games model of PCF [14]. More precisely, we define a

different "linear decomposition" of this model into a category in which morphisms are truly linear, rather than affine. We also ignore the requirement of even-prefix-closure on strategies — this does not change the denotation of programs in the model, nor its full abstraction property.

Definition 3. *The* arena *for a game A is a labelled, bipartite directed acyclic graph, given as a tuple $(M_A, M_A^I, \vdash_A, \lambda_A)$ — where M_A is a set of moves (nodes), $M_A^I \subseteq M_A$ is a specified set of* initial moves *(source nodes), $\vdash_A \subseteq M_A \times (M_A \backslash M_A^I)$ is the enabling (edge) relation and $\lambda_A : M_A \to \{O, P\} \times \{Q, A\}$ is a function partitioning the moves between Player or Opponent, and labelling them as either questions or answers, such that initial moves belong to Opponent and each answer is enabled by a question.*

A *justified sequence* s over A is a sequence over M_A, together with a pointer from each non-initial move b in s to some preceding move a in s such that $a \vdash b$. The set L_A of *legal sequences* over A consists of alternating justified sequences s on A which satisfy *visibility* and *well-bracketing* as defined in [1]. (Details are omitted as nothing here depends on thiese particular conditions, which may be relaxed or modified in various ways to model different combinations of computational effects.) A *game* A is a pair (G_A, P_A) of an arena G_A and a set of justified sequences $P_A \subseteq L_A$. The key constructions are:

- $A \otimes B = (G_A \uplus G_B, \{s \in L_{A \uplus B} \mid s\restriction A \in P_A \wedge s\restriction B \in P_B\})$, where $G_A \uplus G_B$ is the disjoint union of arenas — $(M_A + M_B, M_A^I + M_B^I, \vdash_A + \vdash B_B, [\lambda_A, \lambda_B])$.
- $A \multimap B = (G_A \multimap G_B, \{s \in L_{A \multimap B} \mid s\restriction A \in P_A \wedge s\restriction B \in P_B\})$, where $G_A \multimap G_B$ is the *graft* of A onto the root nodes of B — $M_A + M_B, \mathtt{inr}(M_B^I), (\vdash_A + \vdash_B) \cup \mathtt{inr}(M_B^I) \times \mathtt{inl}(M_A^I), [\overline{\lambda_A}, \lambda_B])$.

3.1 Coherence Space Enrichment of Games

We will now define a **CSpace**-category of games, for which the underlying category is similar to that described in [1] etc. The fact that the inclusion order on strategies provides the latter with an enrichment over the category of cpos and continuous functions was already used in [1], as in other games models, to construct fixed points. Our results amount to showing that strategies form a dI-domain under inclusion, and composition is a bilinear and stable function. Enrichment of a category of games with coherence spaces, or similar categories, is also implicit in earlier work, such as the definition in [5] of a monoidal functor from a similar category of games into a category of ordered coherence spaces. However, this depends on a number of particular features — notably the reconstruction of a strategy on $A \multimap B$ from its projections on A and B, which is not always possible, so **CSpace**-enrichment may be seen as a more general property.

For any game A let $\mathsf{Coh}(A)$ be the coherence space $(P_A^E, \frown_A)$, where P_A^E is the set of even-length sequences in P_A and $s \frown_A t$ if their greatest common prefix $s \sqcap t$ is even length. A strategy on A is a morphism from I to $\mathsf{Coh}(A)$ in **CSpace**, corresponding to a clique of $\mathsf{Coh}(A)$ — an even-branching subset of P_A^E. We define a **CSpace**-category in which objects are games, and the coherence space of morphisms from A to B is $\mathsf{Coh}(A \multimap B)$.

Definition 4. *For $S \subseteq P_{A \multimap B}$ and $T \subseteq P_{B \multimap C}$, let $S|T$ be the set of sequences u on $A + B + C$ such that $u\restriction A \multimap B \in S$ and $u\restriction B \multimap C \in T$. Then* $\mathsf{comp}_{A,B,C} = \{((r,s),t) \in |\mathsf{Coh}(A \multimap B)| \times |\mathsf{Coh}(B \multimap C)| \times |\mathsf{Coh}(A \multimap C)| \mid \exists u \in \{r\}|\{s\}.u\restriction A \multimap C = t\}$.

The identity $\mathsf{id}_A : I \to \mathbf{CSpace}(A \multimap A)$ is the copycat strategy on A consisting of the sequences on $P_{A \multimap A}$ for which each even prefix projects to the same (legal) sequence on both components.

Given strategies $\sigma : I \to \mathsf{Coh}(A \multimap B)$ and $\tau : I \to \mathsf{Coh}(B \multimap C)$, let $\sigma;\tau$ be the relational composition of $\sigma \otimes \tau$ with $\mathsf{comp}_{A,B,C}$. This corresponds to the parallel composition plus hiding of strategies defined in [1], and so by well-pointedness of **CSpace**, satisfies the diagrams for associativity and identity. So it remains to show that comp is stable, for which we require a further key fact about composition — for any $t \in \sigma;\tau$, the "interaction sequence" in $\sigma|\tau$ which restricts to t is unique. This is essentially a version of the "zipping lemma" of [2].

Lemma 3. $\mathsf{comp}_{A,B,C}$ *is a clique of* $\mathsf{Coh}(A \multimap B) \otimes \mathsf{Coh}(B \multimap C) \multimap \mathsf{Coh}(A \multimap C)$.

We define **CSpace**-enriched symmetric monoidal (closed) structure on $\mathbf{G}$, given by the operation $\otimes$ with unit I (the game over the empty arena, with $P_I = \{\varepsilon\}$) and the morphism $\mathsf{tensor}_{A,B,C,D} = \{((r,s),t) \in |\mathsf{Coh}(A \multimap C) \otimes \mathsf{Coh}(B \multimap D)| \times |\mathsf{Coh}(A \otimes B \multimap C \otimes D)| \mid r = t\restriction A \multimap C \wedge s = t\restriction B \multimap D\}$. This corresponds to the action of $\otimes$ on the underlying category of games defined in [1], giving associator, unitor and twist maps making the relevant diagrams commute. The (natural) isomorphism $\mathsf{Coh}(A \otimes B, C) \cong \mathsf{Coh}(A, B \multimap C)$ in **CSpace** yields symmetric monoidal closure.

Note that unlike [1] and many similar models, the underlying symetric monoidal category of games is not *affine* — the unit for the tensor is not a terminal object — there are two morphisms from I to itself, one empty, the other containing the empty sequence. This is a necessary consequence of **CSpace**-enrichment.

For any complete semiring $\mathcal{R}$, change of base yields a symmetric monoidal closed category $\mathbf{G}^{\mathcal{R}} \triangleq \Phi^{\mathcal{R}}_*(\mathbf{G})$ enriched in $\mathbf{Set}_{\mathcal{R}}$. Concretely, a morphism $\phi : A \to B$ in $\mathbf{G}^{\mathcal{R}}_0$ is a *$\mathcal{R}$-weighted strategy* — a map from even-length plays on $A \multimap B$ into $\mathcal{R}$. These are composed by setting

$$(\phi;\psi)(t) = \Sigma\{\phi(u\restriction A \multimap B) \cdot \psi(u\restriction B \multimap C) \mid u \in L_{A \multimap B}|L_{B \to C} \wedge u\restriction A \multimap C = t\}$$

The tensor product of $\mathcal{R}$-weighted strategies $\phi : A \to C, \psi : B \to D$ is $(\phi \otimes \psi)(s) = \phi(s\restriction A \multimap C) \cdot \psi(s\restriction B \multimap D)$.

The faithful, identity-on-objects, monoidal functor $\Phi^{\mathcal{R}}_0 : \mathbf{G}_0 \to \mathbf{G}^{\mathcal{R}}_0$ sends each deterministic strategy $\sigma : A \to B$ to the $\mathcal{R}$-weighted strategy $\sigma_{\mathcal{R}}$ with $\Phi^{\mathcal{R}}_0(\sigma)(s) = 1$ if $s \in \sigma$ and $\Phi^{\mathcal{R}}_0(\sigma)(s) = 0$ otherwise.

By choosing particular semirings we may relate this category to examples in the literature. For instance, if $\mathcal{R}$ is the two-element Boolean ring then morphisms in $\mathbf{G}^{\mathcal{R}}_0$ correspond to sets of legal sequences — i.e. non-deterministic strategies, as in the model of may-testing studied in [13,19].

If $\mathcal{R}$ is the *probability semiring*, $(\mathbb{R}_+^\infty, \Sigma, \cdot, 1)$, then $\mathcal{R}$-weighted strategies correspond precisely to the "probabilistic pre-strategies", introduced by Danos and Harmer [8]. These are refined further by imposing more specfic constraints, although the precursor model is already fully abstract for Probabilistic Algol.

If $\mathcal{R}$ is the *tropical semiring* $(\mathbb{N}^\infty, \bigcup, +, 0)$ then $\mathbf{G}_0^{\mathcal{R}}$ corresponds to a sequential version of Ghica's category of *slot games* [10]. This was introduced as a model quantifying resources used in stateful and concurrent computation, in a presentation rather different to weighted strategies, but equivalent to it. Assuming a distinguished token (\$), which does not occur in the set of moves of any arena, we may define a sequence with costs on the game A to be an interleaving of a sequence $s \in P_A$ of A with a sequence of (\$) moves: a *strategy-with-costs* on a A is a set of such sequences. Strategies with costs $\sigma : A \to B$ and $\tau : B \to C$ are composed by parallel composition with hiding of moves in B, so that all slot moves of σ and τ propagate to $\sigma; \tau$. Taking the weight of a sequence with costs to be the number of slot moves it contains, this is equivalent to the notion of composition for $\mathbb{T}$-weighted strategies — i.e. the category of strategies with costs is isomorphic to the category $\mathbf{G}_{\mathbf{T}}$. The original category of slot games defined in [10] is based on a category (an interleaving model of concurrency, without the alternation condition) which does not, in fact enrich over the category of coherence spaces but does enrich over the category of relations. Thus we may change the base of this model only to free semimodules over an idempotent semiring — of which the tropical semiring is an instance.

4 Additives and Exponentials

We now describe how this change of base can be used to obtain a quantitative semantics of higher-order computation — specifically, an (intuitionistic) *Lafont category* [16] (a symmetric monoidal closed category $\mathcal{C}$ with a "free exponential") with countable biproducts. This notion of categorical model was shown in [18] to yield an adequate model of PCF extended with $\mathcal{R}$-module structure, as described in Sect. 5.

A category $\mathcal{C}$ has set-indexed *biproducts* if it has all set-indexed products and coproducts, and these are *naturally isomorphic* — i.e. for any family J, there is a natural isomorphism (which we may assume to be the identity) between the J-indexed functors $\Pi_{j \in J} -$ and $\coprod_{j \in J}$. Any category with infinite biproducts bears an enrichment over the category of complete monoids and their homomorphisms: Given a family of morphisms $\{f_j : A \to B \mid j \in J\}$, let $\Sigma_{j \in J} f_j = \Delta_A^J; \bigoplus_{j \in J} f_j; \nabla_A^J$, where $\Delta_J^A : A \to \bigoplus_{j \in J} A$ and $\nabla_A^J : \bigoplus_{j \in J} A \to A$ are the diagonal and co-diagonal. Jf $\mathcal{C}$ is a symmetric monoidal closed category, then the tensor *distributes* over biproducts — i.e. $(\bigoplus_{i \in J} A_j) \otimes B = \bigoplus_{j \in J} (A_j \otimes B)$, as they are colimits. The complete monoid enrichment thus extends to the monoidal structure — i.e. the tensor is an enriched functor. Moreover, the endomorphisms on the unit I form a complete, commutative semiring $\mathcal{R}_C = (\mathcal{C}(I, I), \Sigma, \otimes, I)$ — the internal semiring of $\mathcal{C}$.

Conversely, any complete monoid enriched category may be *completed* with biproducts.

Definition 5. *If $\mathcal{C}$ is complete monoid enriched, let $\mathcal{C}^{\Pi}$ be the category in which objects are set-indexed families of objects of $\mathcal{C}$, and morphisms from $\{A_i \mid i \in I\}$ to $\{B_j \mid j \in J\}$ are $I \times J$-indexed sets of morphisms $\{f_{i,j} : A_i \to B_j \mid \langle i, j\rangle \in I \times J\}$, composed by setting $(f; g)_{ik} = \Sigma_{j\in J}(f_{ij}; g_{jk})$.*

If $\mathcal{C}$ is a complete monoid enriched symmetric monoidal category, then we may define the (distributive) tensor product on $\mathcal{C}^{\Pi}$: $\{A_i \mid i \in I\} \otimes \{B_j \mid j \in J\} = \{A_i \otimes B_j \mid \langle i, j\rangle \in I \times J\}$, with $(f \otimes g)_{ikjl} = f_{ij} \otimes g_{kl}$.

Note that the biproduct completion of a complete commutative semiring $\mathcal{R}$ (regarded as a one-object SMCC) is the category $\mathbf{Set}_{\mathcal{R}}$.

4.1 The Cofree Commutative Comonoid

A cofree commutative comonoid on an object B in a symmetric monoidal category $\mathcal{C}$ is an object $(!B, \delta, \epsilon)$ in $\mathsf{comon}(\mathcal{C})$ (the category of commutative commonoids and comonoid morphisms of $\mathcal{C}$) with a (natural in A) isomorphism between $\mathcal{C}(A, B)$ and $\mathsf{comon}(\mathcal{C})(A, !B)$ for each commutative comonoid $(A, \delta_A, \epsilon_A)$. Thus $\mathcal{C}$ has (all) cofree commutative comonoids if and only if the forgetful functor from $\mathsf{comon}(\mathcal{C})$ into $\mathcal{C}$ has a right adjoint. This (monoidal) adjunction resolves a monoidal comonad (the *free exponential*) $! : \mathcal{C} \to \mathcal{C}$.

How can we relate the free exponential to change of base? In general, the category of comonoids of a $\mathcal{V}$-category is not itself $\mathcal{V}$-enriched, so there is no $\mathcal{V}$-adjunction giving rise to a $\mathcal{V}$-enriched free exponential. Moreover, although the categories **CSpace** and $\mathbf{Set}_{\mathcal{R}}$ possess cofree commutative comonoids, these are not preserved by $\Phi^{\mathcal{R}}$ (see discussion below). Instead, we describe properties of a **CSpace**-enriched category which allow the construction of a free exponential on the underlying categories, and their preservation by the functor $\Phi_0^{\mathcal{R}}$. These are based on the existence of *symmetric tensor powers*:

Definition 6. *A family of objects $\{B^i \mid i \in \mathbb{N}\}$ in a symmetric monoidal category are symmetric tensor powers of B if:*

- *For each n there is a morphism $\mathsf{eq}_n : B^n \to B^{\otimes n}$ such that (B^n, eq_n) is an* equalizer *for the group G of automorphisms on $B^{\otimes n}$ derived from the permutations on $\{1, \ldots, n\}$.*
- *These equalizers are preserved by the tensor product — i.e. $(B^m \otimes B^n, \mathsf{eq}_m \otimes \mathsf{eq}_n)$ is an equalizer for the products of pairs of permutation automorphisms.*

The category of coherence spaces has equalizers for any group G of automorphisms on an object D. Let $\sim_G$ be the equivalence relation on $|D|$ induced by G — i.e. $d \sim_G d'$ if there exists $g \in G$ with $(d, d') \in g$. Then the equalizer for G in **CSpace** is the coherence space consisting of those equivalence classes of $\sim_G$ in which all members are coherent — $\{[d]_G \in D/\sim_G \mid d \sim_G d' \implies d \frown_D d'\}$ — with coherence $[d] \frown_E [d']$ if there exists $d'' \in |D|$ such that $d \sim_G d''$.

The category of sets and weighted relations also has equalizers for automorphism groups — in this case given by the set of all equivalence classes of $\sim_G$. Say that an automorphism group $G : D \Rightarrow D$ in **CSpace** is *coherent* whenever $\sim_G \subseteq \frown_A$.

Proposition 1. *$\Phi^{\mathcal{R}}$ preserves the equalizer of G if and only if it is coherent.*

Note that unless $\supset_D = |D| \times |D|$, the group of permutations on $D^{\otimes n}$ in **CSpace** is not coherent — e.g. if $d \not\supset_D d'$ then $(d, d') \not\supset_{A^{\otimes 2}} (d', d)$. So Φ^R does not preserve symmetric tensor powers (and for essentially the same reason does notin general preserve cofree commutative comonoids).

Given an automorphism group $G : A \Rightarrow A$ in a **CSpace**-category, and object B, let $h^B(G) = \{h^B(g) : \mathcal{C}(B, A) \to \mathcal{C}(B, A) \mid g \in G\}$ be the group of automorphisms on $\mathcal{C}(A, B)$ in **CSpace** induced by Yoneda embedding. Say that G is coherent if $h^B(G)$ is coherent for every B.

For instance, in our category of games the group of permutations on $A^{\otimes n}$ is coherent: an atom of $\mathsf{Coh}(B \multimap A^{\otimes n})$ is a justified sequence over $M_B \uplus (M_A \times \{1, \ldots, n\})$, and the equivalence on these sequences induced by the permutation isomorphisms on $A^{\otimes n}$ is simply that induced by permuting the tags on moves in A. This is coherent because it is Opponent who always plays the first move with any given tag.

Proposition 2. *$\Phi_0^{\mathcal{R}} : \mathcal{C}_0 \to \mathcal{C}_0^{\mathcal{R}}$ preserves equalizers for coherent groups.*

Proof. Suppose $(E, \mathsf{eq} : E \to A)$ is an equalizer for a group $G : A \Rightarrow A$. Evidently, $\Phi_0^{\mathcal{R}}(\mathsf{eq}); \Phi_0^{\mathcal{R}}(g) = \Phi_0^{\mathcal{R}}(\mathsf{eq})$ for all $g \in G$ so it remains to show the universal property. Let B be any object of $\mathcal{C}_0^{\mathcal{R}}$ (thus an object of $\mathcal{C}_0$). Then $h^B(\mathsf{eq}); h^B(g) = h^B(g)$ for all $g \in G$ and for any $f : I \to \mathcal{C}(B, A)$ in **CSpace** such that $f; h^B(g) = f$ for all $g \in G$, there exists a unique morphism $u : I \to \mathcal{C}(B, E)$ such that $u; h^B(\mathsf{eq}) = f$. Observe (by well-pointedness of **CSpace**) that this implies that $(h^B(E), h^B(\mathsf{eq}))$ is the equalizer for $h^B(G)$ in **CSpace**.

Thus $(\Phi^{\mathcal{R}}(h^B(E)), \Phi^{\mathcal{R}}(h^B(\mathsf{eq})))$ is an equalizer for $\Phi^{\mathcal{R}}(h^B(G))$ in $\mathbf{Set}_{\mathcal{R}}$, and so for any $f : B \to E$ in $\Phi_*^{\mathcal{R}}(\mathcal{C})$, there exists a unique morphism $u : B \to E$ such that $u; \mathsf{eq} = f$ as required.

If our $\mathbf{Set}_{\mathcal{R}}$-enriched model possesses symmetric tensor powers and infinite, distributive biproducts, this is sufficient to obtain the free exponential as the biproduct of all symmetric tensor powers of B (the *Lafont exponential*) — i.e. $!B = \bigoplus_{n \in \mathbb{N}} B^n$, which is equipped with commutative comonoid structure by defining $\epsilon_{!B} : !B \to I = \pi_0$ and $\delta_{!B} : !B \to !B \otimes !B = \langle \pi_{m+n}; \delta_{m,n} \mid m, n \in \mathbb{N} \rangle$, where $\delta_{m,n} : B_{m+n} \to B^m \otimes B^n$ is the unique morphism such that $\mathsf{eq}_{m+n} = \delta_{m,n}; (\mathsf{eq}_m \otimes \mathsf{eq}_n)$.

Proposition 3. *If $\mathcal{C}$ has symmetric tensor powers, then its biproduct completion $\mathcal{C}^{\Pi}$ has symmetric tensor powers.*

Proof. For $A = \{A_i \mid i \in I\}$, $A^n = \{A_X \mid X \in \mathcal{M}_n(I)\}$, where if X has *support* $i_1, \ldots, i_k$ then $A_X = A_{i_1}^{X(1)} \otimes \ldots \otimes A_{i_k}^{X(i_k)}$.

Thus, given any **CSpace**-enriched category with symmetric tensor powers and consistent permutation groups, we may obtain a Lafont category by changing its base to $\mathbf{Set}_{\mathcal{R}}$, and completing with biproducts. In the case of the **CSpace**-category of games, **G**, we have already argued that the group of permutations

on each tensor power $A^{\otimes n}$ is consistent, and therefore change of base preserves symmetric tensor powers, if they exist. One way to define them is by decomposing the tensor product in $\mathbf{G}$ using the *sequoid* [6,17]. The sequoid $A \oslash B$ is the game $(G_A \uplus G_B, \{t \in P_{A\otimes B} \mid \forall s \sqsubseteq t.s\restriction B = \varepsilon \implies s\restriction A = \varepsilon\})$ — i.e. it is a subgame of $A \otimes B$ in which the first move (if any) is always in A. Let A^n be the n-fold sequoid on A — i.e. $A^0 = I$, $A^{n+1} = A \oslash A^n$, so that plays in A consist of n interleaved plays of A, opened in a fixed order.

Proposition 4. *A^n is an n-ary symmetric tensor power.*

This is established using the categorical structure underlying the sequoid — it is a *monoidal action* of $\mathcal{C}$ upon its subcategory of strict morphisms (strategies on $A \multimap B$ such that every opening move in A is followed by a move in B) — and the fact that the tensor decomposes into the sequoid ($A \otimes B$ is the cartesian product of $A \oslash B$ and $B \oslash A$).[1]

4.2 Preservation of Cofree Commutative Comonoids

In order to lift the functor between underlying categories which is implicit in the change of base to a functor of CCCs between the co-Kleisli categories of the free exponential (preserving the meaning of types and terms in the λ-calculus) it is necessary for it to preserve cofree commutative comonoids. We give conditions for this to hold based on the construction of the latter described in [22] — i.e. (putting it in a nutshell) as the limit of the diagram:

$$I \overset{p_0}{\leftarrow} A_\bullet \overset{p_1}{\leftarrow} A_\bullet^2 \dots A_\bullet^i \overset{p_i}{\leftarrow} (A_\bullet)^{i+1} \dots$$

where $A_\bullet$ is the product $A \times I$ (more precisely, the "free pointed object" on A) and $p_i : (A \times I)^{i+1} \to (A \times I)^i$ is the unique morphism given by the universal property of the symmetric tensor power such that $p_i; \mathsf{eq}_i : A\bullet^{i+1} \to \bullet^{\otimes i} = \mathsf{eq}_{i+1}; (A_\bullet^{\otimes i} \otimes \pi_r)$.

This is a refinement of Lafont's construction (in categories with biproducts, the above limit is $\bigoplus_{i\in\mathbb{N}} A^i$). To show that it is preserved by $\Phi^{\mathcal{R}}_*$, we make the further assumption that for each p_i there is a corresponding morphism $e_i : A_\bullet^i \to A_\bullet^{i+1}$, forming an embedding projection (e-p) pair $(e_i, p_i) : A_i \trianglelefteq B_i$ — i.e. $e_i; p_i = \mathsf{id}_{A_\bullet^i}$ and $p_i; e_i \leq \mathsf{id}_{A^{i+1}}$.

Given an e-p pair from D to E in the category of coherence spaces (which corresponds to a coherence preserving injection from $|D|$ into $|E|$), define $p_\bullet \subseteq |E_\bullet| \times |D_\bullet|$ by $\{(\mathtt{inl}(e), \mathtt{inl}(d)) \mid (d, e) \in p\} \cup \{(e, \mathtt{inr}(*)) \mid e = \mathtt{inr}(*) \vee \nexists d \in D.(d, e) \in p\}$.

CSpace has limits for any chain of such pairs $D_0 \overset{e_0,p_0}{\trianglelefteq} D_1 \overset{e_1,p_1}{\trianglelefteq} \dots$ $|\bigsqcup D| = \{x \in \Pi_{i<\omega}|(D_i)_\bullet| \mid \exists i.x_i \neq \mathtt{inr}(*) \wedge \forall i \in \omega.(x_{i+1}, x_i) \in (p_i)_\bullet\}$, with $x \supset\!\!\circ_{\bigsqcup D} y$ if $x_i \supset\!\!\circ y_i$ for all i. This is also the limit for $D_0 \overset{\Phi^{\mathcal{R}}(e_0),\Phi^{\mathcal{R}}(p_0)}{\trianglelefteq} D_1 \overset{\Phi^{\mathcal{R}}(e_1),\Phi^{\mathcal{R}}(p_1)}{\trianglelefteq} \dots$ in

[1] This structure may all be be given in enriched form, and is therefore preserved by change of base.

$\mathbf{Set}_{\mathcal{R}}$ — i.e. $\Phi^{\mathcal{R}}$ preserves limits for all e-p chains. As in the case of equalizers, we can use this fact, to show that they are preserved by $\Phi_0^{\mathcal{R}}$.

Proposition 5. *If $\bigsqcup A$ is a limit for the chain $A_0 \overset{e_0,p_0}{\trianglelefteq} A_1 \overset{e_1,p_1}{\trianglelefteq} \dots$ in a* **CSpace***-category $\mathcal{C}$, then it is a limit in $\mathcal{C}^{\mathcal{R}}$ for the chain $A_0 \overset{\Phi(e_0,p_0)}{\trianglelefteq} A_1 \overset{\Phi^{\mathcal{R}}(e_1,p_1)}{\trianglelefteq} \dots$*

Proof. The universal property is established as in Prop. 2 — for any B, $\mathcal{C}(B, \bigsqcup A)$ is a limit in **CSpace** for the e-p chain $\mathcal{C}(B, A_0) \trianglelefteq \mathcal{C}(B, A) \trianglelefteq \dots$, and thus $\Phi^{\mathcal{R}}(\mathcal{C}(B, \bigsqcup A))$ is a limit for $\Phi^{\mathcal{R}}(\mathcal{C}(B, A_0)) \trianglelefteq \Phi^{\mathcal{R}}(\mathcal{C}(B, A_2)) \trianglelefteq \dots$, and hence $\bigsqcup A$ is a limit for $A_0 \trianglelefteq A_1 \trianglelefteq \dots$ in $\mathcal{C}^{\mathcal{R}}$.

Any cartesian product in $\mathcal{C}_0$ is a cartesian product in $\mathcal{C}_0^{\mathcal{R}}$ (since $\Phi^{\mathcal{R}}$ preserves products) and so in particular $A \times I$ is the free pointed object on A in $\mathcal{C}_0^{\mathcal{R}}$. Hence, if ! is a limit for $I \overset{p_0}{\leftarrow} A_\bullet \overset{p_1}{\leftarrow} \dots A_\bullet^i \overset{p_2}{\leftarrow} \dots$ in $\mathcal{C}_0$ then it is a limit for $I \overset{\Phi^{\mathcal{R}}(p_0)}{\leftarrow} A_\bullet \overset{\Phi^{\mathcal{R}}(p_1)}{\leftarrow} \dots$ in $\mathcal{C}_0^{\mathcal{R}}$.

As we have observed, our category of games $\mathbf{G}_0$ has all symmetric tensor powers. We also have an embedding-projection pair from A^n to A^{n+1} for each n — viz $e_0 = \bot_{I,A}$, $e_{n+1} = A \oslash e_n$.

G does not have all products, however. In particular, the free pointed object $A \times I$ does not exist in general — e.g. we may show that there is no object $I \times I$ such that $\mathbf{G}(I, I \times I) \cong \mathbf{G}(I, I) \times \mathbf{G}(I, I)$. However we may identify a full subcategory of **G** (i.e. a collection of objects — the *well-opened games*) for which products exist. A game A is *well-opened* if P_A consists only of sequences containing exactly one initial move. If A and B are well-opened then their product $A \times B$ consists of the well-opened sequences in $A \otimes B$. Moreover, if A is well-opened then the free pointed object on A is $(G_A, P_A \cup \varepsilon)$, and the limit $!A$ for the chain $I \overset{p_0}{\leftarrow} A_\bullet \overset{p_1}{\leftarrow} \dots$ is the game consisting of all legal interleavings of sequences in P_A. Note also that if B is well opened then $!A \multimap B$ is well-opened. Hence the "co-Kleisli" category $\mathbf{G}_!$ in which objects are well-opened games, and morphisms from A to B are morphisms from $!A$ to B in **G** is Cartesian closed. This is equivalent to the cartesian closed category of games constructed in [1], etc. (less the even-prefix-closure condition on strategies). Since $\Phi_0^{\mathcal{R}}(!A)$ is the cofree commutative comonoid in $\mathbf{G}_0^{\mathcal{R}}$ for each well-opened game, we have a cartesian closed category of well-opened games and $\mathcal{R}$-weighted strategies, $\mathbf{G}_!^{\mathcal{R}}$ with a cartesian closed functor $\Phi_!^{\mathcal{R}} : \mathbf{G}_! \to \mathbf{G}_!^{\mathcal{R}}$.

5 $\mathcal{R}$-Weighted Idealized Algol

By the results in [1] we know that $\mathbf{G}_!$ furnishes a semantics of Reynolds' *Idealized Algol* — an applied, simply-typed λ-calculus which may be considered as an extension of PCF with integer state (conservative with respect to the operational semantics). So applying the functor $\Phi_0^{\mathcal{R}} : \mathbf{G}_0 \to \mathbf{G}_0^{\mathcal{R}}$ gives us a semantics of Idealized Algol in $\mathbf{G}_!^{\mathcal{R}}$. Since the biproduct completion of $\mathbf{G}_0^{\mathcal{R}}$ is an example of

the categorical model described in [18] (a Lafont category with biproducts) we also have a semantics of $\mathcal{R}$-weighted PCF in $\mathbf{G}_!^{\mathcal{R}}$. This agrees with the semantics of Idealized Algol on their common part (the operations and constants of PCF) and so we may combine both models, to give an interpretation of $\mathrm{IA}^{\mathcal{R}}$ — erratic Idealized Algol with scalar weights from $\mathcal{R}$. Moreover — unlike the weighted relational semantics of $\mathrm{PCF}^{\mathcal{R}}$ — this model is *fully abstract*, a property it inherits directly from the qualitative version.

Types of Idealized Algol are formed from ground types `nat`, `com` (commands) and `var` (integer references). Terms are formed by extending the λ-calculus with fixed points, with the following constants[2]:

- Arithmetic, conditionals: $\mathtt{0} : \mathtt{nat}$, $\mathtt{succ}, \mathtt{pred} : \mathtt{nat} \to \mathtt{nat}$, $\mathtt{Ifz} : \mathtt{nat} \to \mathtt{nat}$.
- Imperative programming: $\mathtt{seq} : \mathtt{com} \to B \to B$ (sequential composition) and $\mathtt{new} : (\mathtt{var} \to B) \to B$ (new variable declaration) where $B \in \{\mathtt{com}, \mathtt{nat}\}$, $\mathtt{assn} : \mathtt{nat} \to \mathtt{var} \to \mathtt{com}$, $\mathtt{deref} : \mathtt{var} \to \mathtt{nat}$, and $\mathtt{mkvar} : \mathtt{nat} \to (\mathtt{nat} \to \mathtt{com}) \to \mathtt{var}$ ("bad variable" construction).
- $\mathcal{R}$-module structure — a nondeterministic choice operator $\mathtt{or} : B \to B \to B$ and scalar multiplication $\mathtt{scl}(\mathtt{k}) : B \to B$ for each $k \in \mathcal{R}$.

The operational semantics for $\mathrm{IA}^{\mathcal{R}}$ extends that given for $\mathrm{PCF}^{\mathcal{R}}$ [20], just as Idealized Algol extends PCF. We define a labelled transition system in which *states* are configurations — pairs $(P, \mathcal{S})$ of a (ground-type) program and a store (a sequence $(a_1, n_1), \ldots, (a_n, n_k)$ of pairs of a location name and integer value). *Labels* are elements of the monoid $(u, a) \in \{l, r\}^* \times \mathcal{R}$ and *actions* take the form $E[M], \mathcal{S} \xrightarrow{u,a} E[M'], \mathcal{S}'$, where $E[\_]$ is an *evaluation context*, given by the grammar: $E ::= [\_] \mid E\ M \mid \mathtt{succ}\ E \mid \mathtt{pred}\ E \mid \mathtt{Ifz}\ E \mid \mathtt{seq}\ E \mid \mathtt{assn}\ E \mid (\mathtt{assn}\ \mathtt{n})\ E \mid \mathtt{deref}\ E$ and $M, \mathcal{S} \xrightarrow{u,a} M', \mathcal{S}'$ is an instance of one of the following rules:

$$
\begin{array}{llll}
(\lambda x.M)\ N, \mathcal{S} & \xrightarrow{\varepsilon,1} M[N/x], \mathcal{S} & \mu x.M, \mathcal{S} & \xrightarrow{\varepsilon,1} M[\mu x.M/x], \mathcal{S} \\
\mathtt{or}, \mathcal{S} & \xrightarrow{l,1} \lambda x.\lambda y.x, \mathcal{S} & \mathtt{or}, \mathcal{S} & \xrightarrow{r,1} \lambda x.\lambda y.y, \mathcal{S} \\
\mathtt{seq}\ \mathtt{skip}, \mathcal{S} & \xrightarrow{\varepsilon,1} \lambda x.x, \mathcal{S} & \mathtt{new}_m\ P, \mathcal{S} & \xrightarrow{\varepsilon,1} P\ a, \mathcal{S}, (a, m) \\
(\mathtt{assn}\ \mathtt{n})\ (\mathtt{mkvar}\ M\ N), \mathcal{S} & \xrightarrow{\varepsilon,1} N\ \mathtt{n}, \mathcal{S} & (\mathtt{assn}\ \mathtt{n})\ a, \mathcal{S} & \xrightarrow{\varepsilon,1} \mathtt{skip}, \mathcal{S}[a_i \mapsto n] \\
\mathtt{Ifz}\ \mathtt{n} + \mathtt{1}, \mathcal{S} & \xrightarrow{\varepsilon,1} \lambda x.\lambda y.y, \mathcal{S} & \mathtt{pred}\ \mathtt{n} + \mathtt{1}, \mathcal{S} & \xrightarrow{\varepsilon,1} \mathtt{n}, \mathcal{S} \\
\mathtt{deref}(\mathtt{mkvar}\ M\ N), \mathcal{S} & \xrightarrow{\varepsilon,1} M, \mathcal{S} & \mathtt{Ifz}\ \mathtt{0}, \mathcal{S} & \xrightarrow{\varepsilon,1} \lambda x.\lambda y.x, \mathcal{S} \\
\mathtt{deref}\ a, \mathcal{S}[(a, n) & \xrightarrow{\varepsilon,1} \mathtt{n}, \mathcal{S} & \mathtt{scl}(\mathtt{k}), \mathcal{S} & \xrightarrow{\varepsilon,k} \lambda x.x, \mathcal{S}
\end{array}
$$

The relation $\xrightarrow{u,a}$ is deterministic, and so we may define the *weight* in $\mathcal{R}$ of each configuration with respect to a sequence $u \in \{l, r\}^*$ of branching choices: $w_u(P, \mathcal{S}) = a$ if $P, \mathcal{S} \xrightarrow{u_1,a_1} \ldots \xrightarrow{u_n,a_n} \mathtt{skip}, \mathcal{S}'$, where $(u, a) = (u_1 \cdot \ldots u_n, a_1 \cdot \ldots \cdot a_n)$, $w_u(P, \mathcal{S}') = 0$ if there is no such sequence of reductions.

The *total weight* of a configuration in $\mathcal{R}$ is given by summing the weights over all possible paths: $w(P, \mathcal{S}) \triangleq \Sigma_{u \in \{l,r\}^*} w_u(P, \mathcal{S})$, and $w(P) = w(P, \_)$.

[2] We consider the variant of IA with *active expressions* and *bad variables* as in [1].

From this notion of testing we derive a notion of equivalence: $P \approx Q$ if for any closing compatible context $C[\_] : \mathtt{com}$, $w(C[P]) = w(C[Q])$.

As discussed in [20] for PCF, the computational meaning of $\mathrm{IA}^{\mathcal{R}}$ depends on the choice of semiring — it may be regarded as a metalanguage for a family of "resource-sensitive" imperative programming languages and their semantics. The weighted games models discussed previously may be viewed as instances of this. Probabilistic games [8] are used to interpret Idealized Algol extended with a constant $\mathtt{coin} : \mathtt{nat}$ which reduces to either $\mathtt{0}$ or $\mathtt{1}$, both with probability 0.5. Thus we may interpret probabilistic Algol inside $\mathrm{IA}^{\mathcal{R}}$ by defining $\mathtt{coin} \triangleq (\mathtt{scl}(0.5)\ \mathtt{0})\ \ \mathtt{or}\ (\mathtt{scl}(0.5)\ \mathtt{1})$.

Slot games are used to give an interpretation of Idealized Concurrent Algol which is sound with respect to an operational semantics which keeps track of the (time, memory, etc.) costs of evaluation as a natural number — each reduction rule is decorated with such a cost, and the worst-case cost is assigned to each program. Setting $\mathcal{R}$ to be the tropical semiring, we may define a translation of non-deterministic IA into $\mathrm{IA}^{\mathcal{R}}$ which is sound with respect to this notion of evaluation, by applying a weighting to each operation corresponding to the cost of its evaluation.

5.1 Denotational Semantics

We interpret $\mathrm{IA}^{\mathcal{R}}$ in the category of games and $\mathcal{R}$-weighted strategies by extending the semantics of $\mathrm{PCF}^{\mathcal{R}}$ [20] with the image under $\Phi_0^{\mathcal{R}}$ of the semantics of the types and constants of Idealized Algol defined in [1]. (We use the version in which each game consists of only *complete* sequences, in which every question is answered.) The types $\mathtt{com}$, $\mathtt{nat}$ and $\mathtt{var}$ denote the (well-opened) games Σ (with a single question and answer) N (with a single Opponent question and Player answers for each value $n \in \mathbb{N}$[3]) and the product $N \times \Sigma^{\omega}$, respectively. The arrow type $S \to T$ denotes the well-opened game $!\llbracket S \rrbracket \multimap \llbracket T \rrbracket$. Terms-in context $x_1 : S_1, \ldots, x :_n S_n \vdash M : T$ are interpreted as morphisms from $\llbracket S_1 \rrbracket \times \ldots \times \llbracket S_n \rrbracket$ to $\llbracket T \rrbracket$ in $\mathbf{G}_!^{\mathcal{R}}$. Each of the constants $C : T$ of Idealized Algol denotes a strategy in $\mathbf{G}_!$, and thus a $\mathcal{R}$-weighted strategy in $\mathbf{G}_!^{\mathcal{R}}$.

This leaves the interpretation of the fixed point operator, which takes a term $\Gamma, x : T \vdash M : T$ to its fixed point $\Gamma \vdash \mu x.M : T$. Semantically, this corresponds to a parameterised fixed point operator on our cartesian closed category of games — a map taking each endomorphism $f \in \mathcal{C}(B \times A, A)$ to a morphism $\mathsf{fix}_B(f) \in \mathcal{C}_!(B, A)$ satisfying $\mathsf{fix}(f) = \langle B, \mathsf{fix}(f) \rangle ; f$. As $\mathbf{G}_!$ is cpo-enriched, this may defined as (parameterised) least fixedpoint. If $\mathcal{R}$ is not *continuously* ordered, then this construction is not available but we may adopt the alternative, described in [18], based on the existence of a *bifree algebra* for the free exponential in $(\mathbf{G}_0^{\mathcal{R}})^{\Pi}$ — the object $\bigoplus_{X \in \mathbb{M}} I$, where $\mathbb{M}$ is the set of nested finite multisets. This is sufficient (cf. [23]) to define a fixed point operator on the co-Kleisli category $\mathbf{G}_!^{\mathcal{R}}$.

[3] In $\mathbf{G}^{\mathcal{R}}$, $N \cong \bigoplus_{i \in \mathbb{N}} \Sigma$.

The *computational adequacy* property for our semantics states that the weight for a program of type com computed by the operational semantics is equal to the weight assigned by its denotation to the single well-opened sequence (qa) in $[\![\mathtt{com}]\!]$. The proof of this follows closely that of [18], defining an equivalent semantics which assigns precise resouce bounds to variables indicating how many times they are called (and in the case of recursively defined variables, a nested finite multiset representing their call-pattern), and proving soundness for this by a nested multiset induction. The only further requirement is a set of equations establishing the soundness of the reduction rules for the constants of Idealized Algol: these were already established in [1] in proving soundness for the semantics in **G**.

Proposition 6. *For every program* $P : \mathtt{com}$, $[\![P, \mathcal{S}]\!](qa) = w(P, \mathcal{S})$.

Although *full abstraction* fails in the semantics of $\mathrm{PCF}^{\mathcal{R}}$ in $\mathbf{Set}_{\mathcal{R}}$ [18], it holds in our model of $\mathrm{IA}^{\mathcal{R}}$, following readily from the *definability property* for the game semantics of Idealized Algol in **G**.

Theorem 1 [1]. *For any finite strategy* $\sigma : [\![T]\!]$ *there exists a IA term* $M_\sigma : T$ *such that* $[\![M_\sigma]\!]_{\mathbf{G}} = \sigma$.

Thus, in particular, every *atomic* strategy on $[\![T]\!]$ (consisting of a single legal sequence) is definable as a term of Idealized Algol (without fixed points), and so any *finitary* strategy in $\mathbf{G}_{\mathcal{R}}$ (i.e. one for which finitely many sequences have non-zero weight) is definable as a finite weighted sum of Idealized Algol terms.

Corollary 1 (Definability for $\mathrm{IA}^{\mathcal{R}}$ **).** *For any finitary* $\mathcal{R}$*-weighted strategy* $\phi : [\![T]\!]$ *there exists a term* $M_\phi : T$ *such that* $[\![M_\phi]\!] = \phi$.

Corollary 2 (Full Abstraction for $\mathrm{IA}^{\mathcal{R}}$ **).** $M \approx M'$ *if and only if* $[\![M]\!] = [\![M']\!]$

Proof. This closely follows the proof in the original model — e.g. for completeness suppose $[\![M]\!] \not\approx [\![M']\!]$, and thus there exists a complete $s \in P_{[\![T]\!]}$ such that $[\![M]\!](s) \neq [\![M']\!](s)$. By the definability property, the strategy $\phi : [\![T]\!] \to [\![\mathtt{com}]\!]$ such that $\phi(t) = 1$ if $t = qsa$ (and 0 otherwise) denotes a term $N : T \to \mathtt{com}$ of Idealized Algol and thus $[\![N\ m]\!]_R(qa) = [\![M]\!](s)$ and $[\![N\ M']\!](qa) = [\![M']\!](s)$. Hence by computational adequacy, $w(N\ M') \neq w(N\ M')$ and so $P \not\approx Q$ as required.

This establishes that any inequivalent terms of $\mathrm{IA}^{\mathcal{R}}$ may be separated by a term of Idealized Algol. So, for example, our model is fully abstract for Probabilistic Algol.

6 Conclusions and Further Directions

We have described a general way of moving from qualitative intensional models to quantitative ones, using the notion of change of base of an enriched category. The only really essential properties that this uses from the original model of

Idealized Algol are that strategies may be viewed as certain cliques in a coherence space and that composition is a stable, linear function, ruling out interleaving models of concurrency, except in the case of idempotent semirings. For the sake of simplicity, we have sidestepped mention of causal order (e.g. prefix order in games), which gives a finer characterization of strategy behaviour. For example, we may enrich categories over event structures [24,25] (or dI-domains [3]) — thus a monoidal functor adding weights to event structures may be used to change their base.

References

1. Abramsky, S., McCusker, G.: Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions. In: O'Hearn, P.W., Tennent, R. (eds.) Algol-like languages. Birkhauser (1997)
2. Baillot, P., Danos, V., Ehrhard, T., Regnier, L.: AJM games are a model of classical linear logic. In: Proceedings of the Twelfth International Symposium on Logic in Computer Science, LICS 1997 (1997)
3. Berry, G.: Stable models of typed λ-calculi. In: Ausiello, G., Böhm, C. (eds.) ICALP 1978. LNCS, vol. 62, pp. 72–89. Springer, Heidelberg (1978). doi:10.1007/3-540-08860-1_7
4. Berry, G., Curien, P.-L.: Sequential algorithms on concrete data structures. Theoret. Comput. Sci. **20**, 265–321 (1982)
5. Calderon, A., McCusker, G.: Understanding game semantics through coherence spaces. Electron. Notes Theoret. Comput. Sci. **265**, 231–244 (2010)
6. Churchill, M., Laird, J., McCusker, G.: Imperative programs as proofs via game semantics. Ann. Pure and Appl. Logic **64**, 27–94 (2013)
7. Cruttwell, G.: Normed Spaces and Change of Base for Enriched Categories. Ph.D. thesis, Dalhousie University (2008)
8. Danos, V., Harmer, R.: Probabilistic game semantics. ACM Trans. Comput. Logic **3**(3), 359–382 (2002)
9. Ehrhard, T.: Hypercoherence: a strongly stable model of linear logic. In: Girard, J.-Y., Lafont, Y., Regnier, L. (eds.) Advances in Linear Logic. Cambridge University Press, Cambridge (1995)
10. Ghica, D.: Slot games: a qunatitative model of computation. In: Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 85–97 (2005)
11. Ghica, D., McCusker, G.: The regular language semantics of second-order Idealised Algol. Theoret. Comput. Sci. **309**, 469–502 (2003)
12. Girard, J.-Y., Taylor, P., Lafont, Y.: Proofs and Types. Cambridge University Press, Cambridge (1990)
13. Harmer, R., McCusker, G.: A fully abstract games semantics for finite nondeterminism. In: Proceedings of the Fourteenth Annual Symposium on Logic in Computer Science, LICS 1999. IEEE Computer Society Press (1998)
14. Hyland, J.M.E., Ong, C.-H.L.: On full abstraction for PCF: I, II and III. Inf. Comput. **163**, 285–408 (2000)
15. Janelidze, G., Kelly, G.M.: A note on actions of a monoidal category. Theor. Appl. Categories **9**(4), 61–91 (2001)
16. Lafont, Y.: Logiques, catégories et machines. Ph.D thesis, Université Paris 7 (1988)

17. Laird J.: A categorical semantics of higher-order store. In: Proceedings of CTCS 2002, number 69 in ENTCS. Elsevier (2002)
18. Laird, J.: Fixed points in quantitative semantics. In: Proceedings of LICS 2016, pp. 347–356. ACM (2016)
19. Laird, J., Manzonetto, G., McCusker, G.: Constructing differential categories and deconstructing categories of games. Inf. Comput. **222**, 247–264 (2013)
20. Laird, J., Manzonetto, G., McCusker, G., Pagani, M.: Weighted relational models of typed lambda-calculi. In: Proceedings of LICS 2013 (2013)
21. Lamarche, F.: Quantitative domains and infinitary algebras. Theoret. Comput. Sci. **94**, 37–62 (1999)
22. Melliès, P.-A., Tabareau, N., Tasson, C.: An explicit formula for the free exponential modality of linear logic. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 247–260. Springer, Heidelberg (2009). doi:10.1007/978-3-642-02930-1_21
23. Simpson, A., Plotkin, G.: Complete axioms for categorical fixed-point operators. In: Proceedings of LICS 2000, pp. 30–41. IEEE Press (2000)
24. Winskel, G.: Event structures. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) ACPN 1986. LNCS, vol. 255, pp. 325–392. Springer, Heidelberg (1987). doi:10.1007/3-540-17906-2_31
25. Winskel, G.: Concurrent strategies. In: Proceedings of LICS 2011 (2011)

Almost Every Simply Typed λ-Term Has a Long β-Reduction Sequence

Ryoma Sin'ya(✉), Kazuyuki Asada, Naoki Kobayashi, and Takeshi Tsukada

The University of Tokyo, Tokyo, Japan
ryoma@kb.is.s.u-tokyo.ac.jp

Abstract. It is well known that the length of a β-reduction sequence of a simply typed λ-term of order k can be huge; it is as large as k-fold exponential in the size of the λ-term in the *worst* case. We consider the following relevant question about *quantitative* properties, instead of the worst case: *how many* simply typed λ-terms have very long reduction sequences? We provide a partial answer to this question, by showing that asymptotically almost every simply typed λ-term of order k has a reduction sequence as long as $(k-2)$-fold exponential in the term size, under the assumption that the arity of functions and the number of variables that may occur in every subterm are bounded above by a constant. The work has been motivated by quantitative analysis of the complexity of higher-order model checking.

1 Introduction

It is well known that the length of a β-reduction sequence of a simply typed λ-term can be extremely long. Beckmann [1] showed that, for any $k \geq 0$,

$$\max\{\beta(t) \mid t \text{ is a simply typed } \lambda\text{-term of order } k \text{ and size } n\} = \mathbf{exp}_k(\Theta(n))$$

where $\beta(t)$ is the maximum length of the β-reduction sequences of the term t, and $\mathbf{exp}_k(x)$ is defined by: $\mathbf{exp}_0(x) \triangleq x$ and $\mathbf{exp}_{k+1}(x) \triangleq 2^{\mathbf{exp}_k(x)}$. Indeed, the following order-k term [1]:

$$(Twice_k)^n\, Twice_{k-1} \cdots Twice_2(\lambda x.a\, x\, x)c,$$

where $Twice_j$ is the twice function $\lambda f^{\sigma_{j-1}}.\lambda x^{\sigma_{j-2}}.f(f\, x)$ (with σ_j being the order-j type defined by: $\sigma_0 = \mathsf{o}$ and $\sigma_j = \sigma_{j-1} \to \sigma_{j-1}$), has a β-reduction sequence of length $\mathbf{exp}_k(\Omega(n))$.

Although the worst-case length of the longest β-reduction sequence is well known as above, much is not known about the *average-case* length of the longest β-reduction sequence: *how often* does one encounter a term having a very long β-reduction sequence? In other words, suppose we pick a simply-typed λ-term t of order k and size n *randomly*; then what is the probability that t has a β-reduction sequence longer than a certain bound, like $\mathbf{exp}_k(cn)$ (where c is some constant)? One may expect that, although there exists a term (like the

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 53–68, 2017.
DOI: 10.1007/978-3-662-54458-7_4

one above) whose reduction sequence is as long as $\mathbf{exp}_k(\Omega(n))$, such a term is rarely encountered.

In the present paper, we provide a partial answer to the above question, by showing that almost every simply typed λ-term of order k has a reduction sequence as long as $(k-2)$-fold exponential in the term size, under a certain assumption. More precisely, we shall show:

$$\lim_{n\to\infty} \frac{\#\left(\{[t]_\alpha \in \Lambda_n^\alpha(k,\iota,\xi) \mid \beta(t) \geq \mathbf{exp}_{k-2}(n)\}\right)}{\#(\Lambda_n^\alpha(k,\iota,\xi))} = 1$$

where $\Lambda_n^\alpha(k,\iota,\xi)$ is the set of (α-equivalence classes $[-]_\alpha$ of) simply-typed λ-terms such that the term size is n, the order is up to k, the (internal) arity is up to $\iota \geq k$ and the number of variable names is up to ξ (see the next section for the precise definition).

To obtain the above result, we use techniques inspired by the quantitative analysis of *untyped* λ-terms [2–4]. For example, David et al. [2] have shown that almost all untyped λ-terms are strongly normalizing, whereas the result is opposite in the corresponding combinatory logic. A more sophisticated analysis is, however, required in our case, for considering only well-typed terms, and also for reasoning about the *length* of a reduction sequence instead of a qualitative property like strong normalization.

This work is a part of our long-term project on the quantitative analysis of the complexity of higher-order model checking [5,6]. The higher-order model checking asks whether the (possibly infinite) tree generated by a ground-type term of the λY-calculus (or, a higher-order recursion scheme) satisfies a given regular property, and it is known that the problem is k-EXPTIME complete for order-k terms [6]. Despite the huge worst-case complexity, practical model checkers [7–9] have been built, which run fast for many typical inputs, and have successfully been applied to automated verification of functional programs [10–13]. The project aims to provide a theoretical justification for it, by studying *how many* inputs actually suffer from the worst-case complexity. Since the problem appears to be hard due to recursion, as an intermediate step towards the goal, we aimed to analyze the variant of the problem considered by Terui [14]: given a term of the simply-typed λ-calculus (without recursion) of type Bool, decide whether it evaluates to true or false (where Booleans are Church-encoded; see [14] for the precise definition). Terui has shown that even for the problem, the complexity is k-EXPTIME complete for order-$(2k+2)$ terms. If, contrary to the result of the present paper, the upper-bound of the lengths of β-reduction sequences *were* small for almost every term, then we could have concluded that the decision problem above is easily solvable for most of the inputs. The result in the present paper does not necessarily provide a negative answer to the question above, because one need not necessarily apply β-reductions to solve Terui's decision problem.

The present work may also shed some light on other problems on typed λ-calculi with exponential or higher worst-case complexity. For example, despite DEXPTIME-completeness of ML typability [15,16], it is often said that the

exponential behavior is rarely seen *in practice*. That is, however, based on only empirical studies. Our technique may be used to provide a theoretical justification (or possibly *un*justification).

The rest of this paper is organized as follows. Section 2 states our main result formally. Section 3 analyzes the asymptotic behavior of the number of typed λ-terms of a given size. Section 4 proves the main result. Section 5 discusses related work, and Sect. 6 concludes the paper. For the space restriction, we omit formal proofs and give only sketches instead; see the full version [17] for details.

2 Main Result

In this section we give the precise statement of our main theorem. We denote the cardinality of a set S by $\#(S)$, and the domain and image of a function f by $\mathrm{Dom}(f)$ and $\mathrm{Im}(f)$, respectively.

The set of (*simple*) *types*, ranged over by τ and σ, is given by: $\tau ::= \mathsf{o} \mid \sigma \to \tau$. Let V be a countably infinite set, which is ranged over by x, x_1, x_2, etc. The set of λ-*terms* (or *terms*), ranged over by t, is defined by:

$$t ::= x \mid \lambda \overline{x}^{\tau}.t \mid t\,t \qquad\qquad \overline{x} ::= x \mid *$$

We call elements of $V \cup \{*\}$ *variables*; $V \cup \{*\}$ is ranged over by $\overline{x}, \overline{x}_1, \overline{x}_2$, etc. We call the special variable $*$ an *unused variable*. We sometimes omit type annotations and just write $\lambda\overline{x}.t$ for $\lambda\overline{x}^{\tau}.t$.

Terms of our syntax can be translated to usual λ-terms by regarding elements in $V \cup \{*\}$ as usual variables. We define the notions of free variables, closed terms, and α-equivalence $\sim_\alpha$ through this identification. The α-equivalence class of a term t is written as $[t]_\alpha$. In this paper, we do not consider a term as an α-equivalence class, and we always use $[-]_\alpha$ explicitly. For a term t, we write $\mathbf{FV}(t)$ for the set of all the free variables of t.

For a term t, we define the set $\mathbf{V}(t)$ of variables (except $*$) in t by:

$$\mathbf{V}(x) \triangleq \{x\} \quad \mathbf{V}(\lambda x^{\tau}.t) \triangleq \{x\} \cup \mathbf{V}(t) \quad \mathbf{V}(\lambda *^{\tau}.t) \triangleq \mathbf{V}(t) \quad \mathbf{V}(t_1 t_2) \triangleq \mathbf{V}(t_1) \cup \mathbf{V}(t_2).$$

Note that neither $\mathbf{V}(t)$ nor even $\#(\mathbf{V}(t))$ is preserved by α-equivalence. For example, $t = \lambda x_1.(\lambda x_2.x_2)(\lambda x_3.x_1)$ and $t' = \lambda x_1.(\lambda x_1.x_1)(\lambda *.x_1)$ are α-equivalent, but $\#(\mathbf{V}(t)) = 3$ and $\#(\mathbf{V}(t')) = 1$.

A *type environment* Γ is a finite set of type bindings of the form $x : \tau$ such that if $(x : \tau), (x : \tau') \in \Gamma$ then $\tau = \tau'$; sometimes we regard an environment also as a function. Note that $(* : \tau)$ cannot belong to a type environment; we do not need any type assumption for $*$ since it does not occur in terms. We give the typing rules as follows:

$$\frac{}{x : \tau \vdash x : \tau} \qquad \frac{\Gamma_1 \vdash t_1 : \sigma \to \tau \qquad \Gamma_2 \vdash t_2 : \sigma}{\Gamma_1 \cup \Gamma_2 \vdash t_1 t_2 : \tau}$$

$$\frac{\Gamma' \vdash t : \tau \qquad \Gamma' = \Gamma \text{ or } \Gamma' = \Gamma \cup \{\overline{x} : \sigma\} \qquad \overline{x} \notin \mathrm{Dom}(\Gamma)}{\Gamma \vdash \lambda \overline{x}^{\sigma}.t : \sigma \to \tau}$$

The above typing rules are equivalent to the usual ones for closed terms, and if $\Gamma \vdash t : \tau$ is derivable, then the derivation is unique. Moreover, if $\Gamma \vdash t : \tau$ then $\mathrm{Dom}(\Gamma) = \mathbf{FV}(t)$. Below we consider only well-typed λ-terms. A pair $\langle \Gamma; \tau \rangle$ of Γ and τ is called a *typing*. We use θ as a metavariable for typings. When $\Gamma \vdash t : \tau$ is derived, we call $\langle \Gamma; \tau \rangle$ a *typing of* a term t, and call t an *inhabitant of* $\langle \Gamma; \tau \rangle$ or a $\langle \Gamma; \tau \rangle$*-term*.

Definition 1 (size, order and internal arity of a term). The *size* of a term t, written $|t|$, is defined by:

$$|\overline{x}| \triangleq 1 \qquad |\lambda \overline{x}^{\tau}.t| \triangleq |t| + 1 \qquad |t_1 t_2| \triangleq |t_1| + |t_2| + 1.$$

The *order* and *internal arity* of a type τ, written $\mathtt{ord}(\tau)$ and $\mathtt{iar}(\tau)$, are defined respectively by:

$$\begin{aligned} \mathtt{ord}(\mathtt{o}) &\triangleq 0 \qquad \mathtt{iar}(\mathtt{o}) \triangleq 0 \\ \mathtt{ord}(\tau_1 \to \cdots \to \tau_n \to \mathtt{o}) &\triangleq \max\{\mathtt{ord}(\tau_i) + 1 \mid 1 \le i \le n\} \qquad (n \ge 1) \\ \mathtt{iar}(\tau_1 \to \cdots \to \tau_n \to \mathtt{o}) &\triangleq \max(\{n\} \cup \{\mathtt{iar}(\tau_i) \mid 1 \le i \le n\}) \quad (n \ge 1). \end{aligned}$$

For a $\langle \Gamma; \tau \rangle$-term t, we define the order and internal arity of $\Gamma \vdash t : \tau$ written $\mathtt{ord}(\Gamma \vdash t : \tau)$ and $\mathtt{iar}(\Gamma \vdash t : \tau)$ by:

$$\begin{aligned} \mathtt{ord}(\Gamma \vdash t : \tau) &\triangleq \max\{\mathtt{ord}(\tau') \mid (\Gamma' \vdash t' : \tau') \text{ occurs in } \Delta\} \\ \mathtt{iar}(\Gamma \vdash t : \tau) &\triangleq \max\{\mathtt{iar}(\tau') \mid (\Gamma' \vdash t' : \tau') \text{ occurs in } \Delta\} \end{aligned}$$

where Δ is the (unique) derivation tree for $\Gamma \vdash t : \tau$.

Note that the notions of size, order, internal arity, and $\beta(t)$ (defined in the introduction) are well-defined with respect to α-equivalence.

Definition 2 (terms with bounds on types and variables). Let $\delta, \iota, \xi \ge 0$ and $n \ge 1$ be integers. We denote by $\mathrm{Types}(\delta, \iota)$ the set of types $\{\tau \mid \mathtt{ord}(\tau) \le \delta, \mathtt{iar}(\tau) \le \iota\}$. For Γ and τ we define:

$$\begin{aligned} \Lambda^{\alpha}_n(\langle \Gamma; \tau \rangle, \delta, \iota, \xi) \triangleq \{[t]_\alpha \mid\ & \Gamma \vdash t : \tau,\ |t| = n,\ \min_{t' \in [t]_\alpha} \#(\mathbf{V}(t')) \le \xi, \\ & \mathtt{ord}(\Gamma \vdash t : \tau) \le \delta,\ \mathtt{iar}(\Gamma \vdash t : \tau) \le \iota\} \\ \Lambda^{\alpha}(\langle \Gamma; \tau \rangle, \delta, \iota, \xi) \triangleq & \bigcup_{n \ge 1} \Lambda^{\alpha}_n(\langle \Gamma; \tau \rangle, \delta, \iota, \xi). \end{aligned}$$

Also we define:

$$\Lambda^{\alpha}_n(\delta, \iota, \xi) \triangleq \bigcup_{\tau \in \mathrm{Types}(\delta, \iota)} \Lambda^{\alpha}_n(\langle \emptyset; \tau \rangle, \delta, \iota, \xi) \qquad \Lambda^{\alpha}(\delta, \iota, \xi) \triangleq \bigcup_{n \ge 1} \Lambda^{\alpha}_n(\delta, \iota, \xi).$$

Our main result is the following theorem, which will be proved in Sect. 4.

Theorem 1. *Let $\delta, \iota, \xi \ge 2$ be integers and let $k = \min\{\delta, \iota\}$. Then,*

$$\lim_{n \to \infty} \frac{\#\big(\{[t]_\alpha \in \Lambda^{\alpha}_n(\delta, \iota, \xi) \mid \beta(t) \ge \mathbf{exp}_{k-2}(n)\}\big)}{\#(\Lambda^{\alpha}_n(\delta, \iota, \xi))} = 1.$$

Remark 1. Note that in the above theorem, the order δ, the internal arity ι and the number ξ of variables are bounded above by a constant, independently of the term size n. It is debatable whether the assumption is reasonable, and a slight change of the assumption may change the result, as is the case for strong normalization of untyped λ-term [2,4]. When λ-terms are viewed as models of functional programs, our rationale behind the assumption is as follows. The assumption that the size of types (hence also the order and the internal arity) is fixed is sometimes assumed in the context of type-based program analysis [18]. The assumption on the number of variables comes from the observation that a large program usually consists of a large number of *small* functions, and that the number of variables is bounded by the size of each function.

3 Analysis of $\Lambda_n^\alpha(\delta, \iota, \xi)$

To prove our main theorem, we first analyze some formal language theoretic structure and properties of $\Lambda^\alpha(\delta, \iota, \xi)$: in Sect. 3.1, we construct a regular tree grammar such that there is a size preserving bijection between its tree language and $\Lambda^\alpha(\delta, \iota, \xi)$; in Sect. 3.2, we show that the grammar has two important properties: irreducibility and aperiodicity. Thanks to those properties, we can obtain a simple asymptotic formula for $\#(\Lambda_n^\alpha(\delta, \iota, \xi))$ using analytic combinatorics [19]. The irreducibility and aperiodicity properties will also be used in Sect. 4 for adjusting the size and typing of a λ-term.

3.1 $\Lambda^\alpha(\delta, \iota, \xi)$ as a Regular Tree Language

We first recall some basic definitions for regular tree grammars. A *ranked alphabet* Σ is a mapping from a finite set of symbols to the set of natural numbers. For a symbol $a \in \mathrm{Dom}(\Sigma)$, we call $\Sigma(a)$ the *rank* of a. A *Σ-tree* is a tree composed from symbols in Σ according to their ranks: (i) a is a Σ-tree if $\Sigma(a) = 0$, (ii) $a(T_1, \cdots, T_{\Sigma(a)})$ is a Σ-tree if $\Sigma(a) \geq 1$ and T_i is a Σ-tree for each $i \in \{1, \ldots, \Sigma(a)\}$. We use the meta-variable T for trees. The *size* of T, written as $|T|$, is the number of nodes and leaves of T. We denote the set of all Σ-trees by $\mathcal{T}_\Sigma$.

A *regular tree grammar* is a triple $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R})$ where (i) Σ is a ranked alphabet; (ii) $\mathcal{N}$ is a finite set of *non-terminals*; (iii) $\mathcal{R}$ is a finite set of *rewriting rules* of the form $N \longrightarrow a(N_1, \cdots, N_{\Sigma(a)})$ where $a \in \mathrm{Dom}(\Sigma)$, $N \in \mathcal{N}$ and $N_i \in \mathcal{N}$ for every $i \in \{1, \ldots, \Sigma(a)\}$. A *$(\Sigma \cup \mathcal{N})$-tree* T is a tree composed from symbols in $\Sigma \cup \mathcal{N}$ according to their ranks where the rank of every symbol in $\mathcal{N}$ is zero (thus non-terminals appear only in leaves of T). For a tree grammar $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R})$ and a non-terminal $N \in \mathcal{N}$, *the language $\mathcal{L}(\mathcal{G}, N)$ of N* is defined by $\mathcal{L}(\mathcal{G}, N) \triangleq \{T \in \mathcal{T}_\Sigma \mid N \longrightarrow_{\mathcal{G}}^* T\}$ where $\longrightarrow_{\mathcal{G}}^*$ denotes the reflexive and transitive closure of the rewriting relation $\longrightarrow_{\mathcal{G}}$. We also define $\mathcal{L}_n(\mathcal{G}, N) \triangleq \{T \in \mathcal{T}_\Sigma \mid N \longrightarrow^* T, |T| = n\}$. We often omit $\mathcal{G}$ and write $N \longrightarrow^* N'$, $\mathcal{L}(N)$, and $\mathcal{L}_n(N)$ for $N \longrightarrow_{\mathcal{G}}^* N'$, $\mathcal{L}(\mathcal{G}, N)$, and $\mathcal{L}_n(\mathcal{G}, N)$ respectively, if $\mathcal{G}$ is clear from the context. We say that N' is *reachable* from N if there exists a $(\Sigma \cup \mathcal{N})$-tree T

such that $N \longrightarrow^* T$ and T contains N' as a leaf. A grammar $\mathcal{G}$ is *unambiguous* if, for every pair of a non-terminal N and a tree T, there exists at most one leftmost reduction sequence from N to T.

Definition 3 (grammar of $\Lambda^\alpha(\delta,\iota,\xi)$). Let $\delta,\iota,\xi \geq 0$ be integers and $X_\xi = \{x_1,\cdots,x_\xi\}$ be a subset of V. The regular tree grammar $\mathcal{G}(\delta,\iota,\xi)$ is defined as $(\Sigma(\delta,\iota,\xi),\mathcal{N}(\delta,\iota,\xi),\mathcal{R}(\delta,\iota,\xi))$ where:

$$\begin{aligned}
\Sigma(\delta,\iota,\xi) &\triangleq \{x \mapsto 0 \mid x \in X_\xi\} \cup \{@ \mapsto 2\} \\
&\quad \cup \{\lambda\overline{x}^\tau \mapsto 1 \mid \overline{x} \in \{*\} \cup X_\xi,\ \tau \in \mathrm{Types}(\delta-1,\iota)\} \\
\mathcal{N}(\delta,\iota,\xi) &\triangleq \{N_{\langle\Gamma;\tau\rangle} \mid \tau \in \mathrm{Types}(\delta,\iota), \mathrm{Dom}(\Gamma) \subseteq X_\xi,\ \mathrm{Im}\,(\Gamma) \subseteq \mathrm{Types}(\delta-1,\iota), \\
&\qquad\qquad \Gamma \vdash t : \tau \text{ for some } t\ \} \\
\mathcal{R}(\delta,\iota,\xi) &\triangleq \{N_{\langle\{x_i:\tau\};\tau\rangle} \longrightarrow x_i\} \cup \{N_{\langle\Gamma;\sigma\to\tau\rangle} \longrightarrow \lambda *^\sigma (N_{\langle\Gamma;\tau\rangle})\} \\
&\quad \cup \{N_{\langle\Gamma;\sigma\to\tau\rangle} \longrightarrow \lambda x_i^\sigma (N_{\langle\Gamma\cup\{x_i:\sigma\};\tau\rangle}) \mid i = \min\{j \geq 1 \mid x_j \notin \mathrm{Dom}(\Gamma)\}, \\
&\qquad \#(\Gamma) < \xi\} \cup \{N_{\langle\Gamma;\tau\rangle} \longrightarrow @(N_{\langle\Gamma_1;\sigma\to\tau\rangle}, N_{\langle\Gamma_2;\sigma\rangle}) \mid \Gamma = \Gamma_1 \cup \Gamma_2\}
\end{aligned}$$

Here, the special symbol $@ \in \mathrm{Dom}(\Sigma(\delta,\iota,\xi))$ corresponds to application. For a technical convenience, the above definition excludes from $\mathcal{N}(\delta,\iota,\xi)$ typings which have no inhabitant. Note that $\Sigma(\delta,\iota,\xi)$, $\mathcal{N}(\delta,\iota,\xi)$ and $\mathcal{R}(\delta,\iota,\xi)$ are finite. To see the finiteness of $\mathcal{N}(\delta,\iota,\xi)$, notice that X_ξ and $\mathrm{Types}(\delta-1,\iota)$ are finite, hence so is $\{\Gamma \mid \mathrm{Dom}(\Gamma) \subseteq X_\xi,\ \mathrm{Im}\,(\Gamma) \subseteq \mathrm{Types}(\delta-1,\iota)\}$. The finiteness of $\mathcal{R}(\delta,\iota,\xi)$ follows immediately from that of $\mathcal{N}(\delta,\iota,\xi)$.

Example 1. Let us consider the case where $\delta = \iota = \xi = 1$. The grammar $\mathcal{G}(1,1,1)$ consists of the following.

$$\Sigma(1,1,1) = \{x_1, @, \lambda x_1^\circ, \lambda *^\circ\} \qquad \mathcal{N}(1,1,1) = \{N_{\langle\emptyset;\mathtt{o}\to\mathtt{o}\rangle}, N_{\langle\{x_1:\mathtt{o}\};\mathtt{o}\rangle} N_{\langle\{x_1:\mathtt{o}\};\mathtt{o}\to\mathtt{o}\rangle}\}$$

$$\mathcal{R}(1,1,1) = \begin{cases} N_{\langle\emptyset;\mathtt{o}\to\mathtt{o}\rangle} \longrightarrow \lambda x_1^\circ (N_{\langle\{x_1:\mathtt{o}\};\mathtt{o}\rangle}) \\ N_{\langle\{x_1:\mathtt{o}\};\mathtt{o}\rangle} \longrightarrow x_1 | @(N_{\langle\{x_1:\mathtt{o}\};\mathtt{o}\to\mathtt{o}\rangle}, N_{\langle\{x_1:\mathtt{o}\};\mathtt{o}\rangle}) | @(N_{\langle\emptyset;\mathtt{o}\to\mathtt{o}\rangle}, N_{\langle\{x_1:\mathtt{o}\};\mathtt{o}\rangle}) \\ N_{\langle\{x_1:\mathtt{o}\};\mathtt{o}\to\mathtt{o}\rangle} \longrightarrow \lambda *^\circ (N_{\langle\{x_1:\mathtt{o}\};\mathtt{o}\rangle}). \end{cases}$$

There is the obvious embedding $e^{(\delta,\iota,\xi)}$ (e for short) from trees in $\mathcal{T}_{\Sigma(\delta,\iota,\xi)}$ into λ-terms. For $N_{\langle\Gamma;\tau\rangle} \in \mathcal{N}(\delta,\iota,\xi)$ we define

$$\pi^{(\delta,\iota,\xi)}_{\langle\Gamma;\tau\rangle} \triangleq [-]_\alpha \circ e : \mathcal{L}\left(N_{\langle\Gamma;\tau\rangle}\right) \to \Lambda^\alpha(\langle\Gamma;\tau\rangle,\delta,\iota,\xi).$$

We sometimes omit the superscript and/or the subscript.

Proposition 1. *For $\delta,\iota,\xi \geq 0$, $\pi_{\langle\Gamma;\tau\rangle}$ is a size-preserving bijection, and $\mathcal{G}(\delta,\iota,\xi)$ is unambiguous.*

The former part of Proposition 1 says that $\mathcal{G}(\delta,\iota,\xi)$ gives a complete representation system of the α-equivalence classes. For $[t]_\alpha \in \Lambda^\alpha(\langle\Gamma;\tau\rangle,\delta,\iota,\xi)$, we define $\nu^{(\delta,\iota,\xi)}_{\langle\Gamma;\tau\rangle}([t]_\alpha)$ (or $\nu([t]_\alpha)$ for short) as $e^{(\delta,\iota,\xi)} \circ \left(\pi^{(\delta,\iota,\xi)}_{\langle\Gamma;\tau\rangle}\right)^{-1}([t]_\alpha)$. The function ν *normalizes* variable names. For example, $t = \lambda x.x(\lambda y.\lambda z.z)$ is normalized to $\nu([t]_\alpha) = \lambda x_1.x_1(\lambda *.\lambda x_1.x_1)$.

Due to technical reasons, we restrict the grammar $\mathcal{G}(\delta,\iota,\xi)$ to $\mathcal{G}^{\emptyset}(\delta,\iota,\xi)$, which contains only non-terminals reachable from $N_{\langle\emptyset;\sigma\rangle}$ for some σ (see the full version [17] for details).

$$\mathcal{N}^{\emptyset}(\delta,\iota,\xi) \triangleq \{N_\theta \in \mathcal{N}(\delta,\iota,\xi) \mid N_\theta \text{ is reachable from some } N_{\langle\emptyset;\sigma\rangle} \in \mathcal{N}(\delta,\iota,\xi)\}$$
$$\mathcal{R}^{\emptyset}(\delta,\iota,\xi) \triangleq \{N_\theta \longrightarrow T \in \mathcal{R}(\delta,\iota,\xi) \mid N_\theta \in \mathcal{N}^{\emptyset}(\delta,\iota,\xi)\}$$
$$\mathcal{G}^{\emptyset}(\delta,\iota,\xi) \triangleq (\Sigma(\delta,\iota,\xi), \mathcal{N}^{\emptyset}(\delta,\iota,\xi), \mathcal{R}^{\emptyset}(\delta,\iota,\xi)).$$

For $N_\theta \in \mathcal{N}^{\emptyset}(\delta,\iota,\xi)$, clearly $\mathcal{L}\left(\mathcal{G}^{\emptyset}(\delta,\iota,\xi), N_\theta\right) = \mathcal{L}\left(\mathcal{G}(\delta,\iota,\xi), N_\theta\right)$. Through the bijection π, we can show that, for any $N_{\langle\Gamma;\tau\rangle} \in \mathcal{N}(\delta,\iota,\xi)$, $N_{\langle\Gamma;\tau\rangle}$ also belongs to $\mathcal{N}^{\emptyset}(\delta,\iota,\xi)$ if and only if there exists a term in $\Lambda^{\alpha}(\delta,\iota,\xi)$ whose derivation contains a type judgment of the form $\Gamma \vdash t : \tau$.

3.2 Irreducibility and Aperiodicity

We discuss two important properties of the grammar $\mathcal{G}^{\emptyset}(\delta,\iota,\xi)$ where $\delta,\iota,\xi \geq 2$: irreducibility and aperiodicity [19].[1]

Definition 4 (irreducibility and aperiodicity). Let $\mathcal{G} = (\Sigma,\mathcal{N},\mathcal{R})$ be a regular tree grammar. We say that $\mathcal{G}$ is:

- *non-linear* if $\mathcal{R}$ contains at least one rule of the form $N \longrightarrow a(N_1,\cdots,N_{\Sigma(a)})$ with $\Sigma(a) \geq 2$,
- *strongly connected* if for any pair of non-terminals $N_1, N_2 \in \mathcal{N}$, N_1 is reachable from N_2,
- *irreducible* if $\mathcal{G}$ is both non-linear and strongly connected,
- *aperiodic* if for any non-terminal $N \in \mathcal{N}$ there exists an integer $m > 0$ such that $\#(\mathcal{L}_n(N)) > 0$ for any $n > m$.

Proposition 2. *$\mathcal{G}^{\emptyset}(\delta,\iota,\xi)$ is irreducible and aperiodic for any $\delta,\iota,\xi \geq 2$.*

The following theorem is a minor modification of Theorem VII.5 in [19], which states the asymptotic behavior of an irreducible and aperiodic *context-free specification* (see the full version [17] for details). Below, $\sim$ means the *asymptotic equality*, i.e., $f(n) \sim g(n) \iff \lim_{n\to\infty} f(n)/g(n) = 1$.

Theorem 2 ([19]). *Let $\mathcal{G} = (\Sigma,\mathcal{N},\mathcal{R})$ be an unambiguous, irreducible and aperiodic regular tree grammar. Then there exists a constant $\gamma(\mathcal{G}) > 1$ such that, for any non-terminal $N \in \mathcal{N}$, there exists a constant $C_N(\mathcal{G}) > 0$ such that*

$$\#(\mathcal{L}_n(N)) \sim C_N(\mathcal{G})\gamma(\mathcal{G})^n n^{-3/2}.$$

As a corollary of Proposition 2 and Theorem 2 above, we obtain:

$$\#(\Lambda_n^{\alpha}(\delta,\iota,\xi)) \sim C\gamma^n n^{-3/2} \tag{1}$$

where $C > 0$ and $\gamma > 1$ are some real constants determined by δ, ι, $\xi \geq 2$. For proving our main theorem, we use a variation of the formula (1) above, stated as Lemma 1 later.

[1] In [19], irreducibility and aperiodicity are defined for *context-free specifications*. Our definition is a straightforward adaptation of the definition for regular tree grammars.

4 Proof of the Main Theorem

We give a proof of Theorem 1 in this section. In the rest of the paper, we denote by $\log^{(2)}(n)$ the 2-fold logarithm: $\log^{(2)}(n) \triangleq \log\log n$. All logarithms are base 2. The outline of the proof is as follows. We prepare a family $(t_n)_{n\in\mathbb{N}}$ of λ-terms such that t_n is of size $\Omega(\log^{(2)}(n))$ and has a β-reduction sequence of length $\mathbf{exp}_k(\Omega(|t_n|))$, i.e., $\mathbf{exp}_{k-2}(\Omega(n))$. Then we show that almost every λ-term of size n contains t_n as a subterm. The latter is shown by adapting (a parameterized version of) the *infinite monkey theorem*[2] for words to simply-typed λ-terms.

To clarify the idea, let us first recall the infinite monkey theorem for words. Let A be an alphabet, i.e., a finite non-empty set of symbols. For a word $w = a_1 \cdots a_n$, we write $|w| = n$ for the length of w. As usual, we denote by A^n the set of all words of length n over A, and by A^* the set of all finite words over A: $A^* = \bigcup_{n\geq 0} A^n$. For two words $w, w' \in A^*$, we say w' is a *subword of* w and write $w' \sqsubseteq w$ if $w = w_1 w' w_2$ for some words $w_1, w_2 \in A^*$. The infinite monkey theorem states that, for any word $w \in A^*$, the probability that a randomly chosen word of size n contains w as a subword tends to one if n tends to infinity.

To prove our main theorem, we need to extend the above infinite monkey theorem to the following parameterized version[3], and then further extend it for simply-typed λ-terms instead of words. We give a proof of the following proposition, because it will clarify the overall structure of the proof of the main theorem.

Proposition 3. (parameterized infinite monkey theorem). *Let A be an alphabet and $(w_n)_n$ be a family of words over A such that $|w_n| = \lceil \log^{(2)}(n) \rceil$. Then, we have:*

$$\lim_{n\to\infty} \frac{\#(\{w \in A^n \mid w_n \sqsubseteq w\})}{\#(A^n)} = 1.$$

Proof. Let $p(n)$ be $1 - \#(\{w \in A^n \mid w_n \sqsubseteq w\})/\#(A^n)$, i.e., the probability that a word of size n does *not* contain w_n. We write $s(n)$ for $\lceil \log^{(2)}(n) \rceil$ and $c(n)$ for $\lfloor n/s(n) \rfloor$. Given a word $w = a_1 \cdots a_n \in A^n$, let us partition it to subwords of length $s(n)$ as follows.

$$w = \underbrace{a_1 \cdots a_{s(n)}}_{\text{1-st subword}} \cdots \underbrace{a_{(c(n)-1)s(n)+1} \cdots a_{c(n)s(n)}}_{c(n)\text{-th subword}} a_{c(n)s(n)+1} \cdots a_n$$

Then,

$$\begin{aligned} p(n) &\leq \text{“the probability that none of the } i\text{-th subword is } w_n'' \\ &= \left(\frac{\#(A^{s(n)} \setminus \{w_n\})}{\#(A^{s(n)})}\right)^{c(n)} = \left(\frac{\#(A^{s(n)}) - 1}{\#(A^{s(n)})}\right)^{c(n)} = \left(1 - \frac{1}{\#(A)^{s(n)}}\right)^{c(n)}. \end{aligned}$$

[2] It is also called as “Borges's theorem” (*cf.* [19, p. 61, Note I.35]).

[3] Although it is a simple extension, we are not aware of literature that explicitly states this parameterized version.

Since $\left(1-\frac{1}{\#(A)^{s(n)}}\right)^{c(n)} = \left(1-\frac{1}{\#(A)^{\lceil \log^{(2)}(n)\rceil}}\right)^{\lfloor n/\lceil \log^{(2)}(n)\rceil \rfloor}$ tends to zero (see the full version [17]) if n tends to infinity, we have the required result. □

To prove an analogous result for simply-typed λ-terms, we consider below subcontexts of a given term instead of subwords of a given word. To consider "contexts up to α-equivalence", in Sect. 4.1 we introduce the set $\mathcal{U}^{\nu}_{n}(\delta,\iota,\xi)$ of "normalized" contexts (of size n and with the restriction by δ, ι and ξ), where $\mathcal{U}^{\nu}_{s(n)}(\delta,\iota,\xi)$ corresponds to $A^{s(n)}$ above, and give an upper bound of $\#(\mathcal{U}^{\nu}_{n}(\delta,\iota,\xi))$. A key property used in the above proof was that any word of length n can be partitioned to sufficiently many subwords of length $\log^{(2)}(n)$. Section 4.2 below shows an analogous result that any term of size n can be decomposed into sufficiently many subcontexts of a given size. Section 4.3 constructs a family of contexts $Expl^{k}_{n}$ (called "explosive contexts") that have very long reduction sequences; $(Expl^{k}_{n})_{n}$ corresponds to $(w_n)_n$ above. Finally, Sect. 4.4 proves the main theorem using an argument similar to (but more involved than) the one used in the proof above.

4.1 Normalized Contexts

We first introduce some basic definitions of contexts, and then we define the notion of a normalized context, which is a context normalized by the function ν given in Sect. 3.1.

The set of *contexts*, ranged over by C, is defined by

$$C ::= [\,] \mid x \mid \lambda\overline{x}^{\tau}.C \mid CC$$

The *size* of C, written $|C|$, is defined by:

$$|[\,]| \triangleq 0 \qquad |x| \triangleq 1 \qquad |\lambda\overline{x}^{\tau}.C| \triangleq |C|+1 \qquad |C_1C_2| \triangleq |C_1|+|C_2|+1.$$

We call a context C an *n-context* (and define $\mathtt{hn}(C) \triangleq n$) if C contains n occurrences of $[\,]$. We use the metavariable S for 1-contexts. A *0/1-context* is a term t or a 1-context S and we use the metavariable u to denote 0/1-contexts. The holes in C occur as leaves and we write $[\,]_i$ for the i-th hole, which is counted in the left-to-right order.

For C, C_1, …, $C_{\mathtt{hn}(C)}$, we write $C[C_1]\dots[C_{\mathtt{hn}(C)}]$ for the context obtained by replacing $[\,]_i$ in C with C_i for each $i \le \mathtt{hn}(C)$. For C and C', we write $C[C']_i$ for the context obtained by replacing the i-th hole $[\,]_i$ in C with C'. As usual, these substitutions may capture variables; e.g., $(\lambda x.[\,])[x]$ is $\lambda x.x$. We say that C is a *subcontext of* C' and write $C \preceq C'$ if there exist C'', $1 \le i \le \mathtt{hn}(C'')$ and $C_1, \cdots, C_{\mathtt{hn}(C)}$ such that $C' = C''[C[C_1]\cdots[C_{\mathtt{hn}(C)}]]_i$.

The set of *context typings*, ranged over by κ, is defined by: $\kappa ::= \theta_1 \cdots \theta_k \Rightarrow \theta$ where $k \in \mathbb{N}$ and θ_i is a typing of the form $\langle \Gamma_i; \tau_i\rangle$ for each $1 \le i \le k$ (recall that we use θ as a metavariable for typings). A $\langle \Gamma_1;\tau_1\rangle \cdots \langle \Gamma_k;\tau_k\rangle \Rightarrow \langle \Gamma;\tau\rangle$-context is a k-context C such that $\Gamma \vdash C : \tau$ is derivable from $\Gamma_i \vdash [\,]_i : \tau_i$. We identify a context typing $\Rightarrow\theta$ with the typing θ, and call a θ-context also a *θ-term*.

From now, we begin to define normalized contexts. First we consider contexts in terms of the grammar $\mathcal{G}^{\emptyset}(\delta,\iota,\xi)$ given in Sect. 3.1. Let $\delta,\iota,\xi \geq 0$. For $\kappa = \theta_1 \cdots \theta_n \Rightarrow \theta$ such that $N_{\theta_1},\ldots,N_{\theta_n},N_\theta \in \mathcal{N}(\delta,\iota,\xi)$, a ($\kappa$-)*context-tree* is a tree $\widehat{T}$ in $\mathcal{T}_{\Sigma(\delta,\iota,\xi)\cup\mathcal{N}(\delta,\iota,\xi)}$ such that there exists a reduction $N_\theta \longrightarrow^* \widehat{T}$ and the occurrences of non-terminals in $\widehat{T}$ (in the left-to-right order) are exactly $N_{\theta_1},\ldots,N_{\theta_n}$. We use $\widehat{T}$ as a metavariable for context-trees. We write $\mathcal{L}(\kappa,\delta,\iota,\xi)$ for the set of all κ-context-trees. For $\theta_1\cdots\theta_n\Rightarrow\theta$-context-tree $\widehat{T}$ and $\theta^i_1\cdots\theta^i_{k_i}\Rightarrow\theta_i$-context-trees $\widehat{T}_i$ $(i=1,\ldots,n)$, we define the *substitution* $\widehat{T}[\widehat{T}_1]\cdots[\widehat{T}_n]$ as the $\theta^1_1\cdots\theta^1_{k_1}\cdots\theta^n_1\cdots\theta^n_{k_n}\Rightarrow\theta$- context-tree obtained by replacing N_{θ_i} in $\widehat{T}$ with $\widehat{T}_i$.

The set $\mathcal{C}^\nu(\kappa,\delta,\iota,\xi)$ of *normalized κ-contexts* is defined by:

$$\mathcal{C}^\nu(\kappa,\delta,\iota,\xi) \triangleq e_\kappa^{(\delta,\iota,\xi)}(\mathcal{L}(\kappa,\delta,\iota,\xi))$$

where $e_\kappa^{(\delta,\iota,\xi)}$ is the obvious embedding from κ-context-trees to κ-contexts that preserves the substitution (i.e., $e_\kappa^{(\delta,\iota,\xi)}(T[T']) = e_\kappa^{(\delta,\iota,\xi)}(T)[e_\kappa^{(\delta,\iota,\xi)}(T')]$). Further, the sets $\mathcal{U}^\nu(\delta,\iota,\xi)$ and $\mathcal{U}^\nu_n(\delta,\iota,\xi)$ of *normalized 0/1-contexts* are defined by:

$$\mathcal{U}^\nu(\delta,\iota,\xi) \triangleq \Big(\bigcup_{N_\theta\in\mathcal{N}^\emptyset(\delta,\iota,\xi)} \mathcal{C}^\nu(\theta,\delta,\iota,\xi)\Big) \bigcup \Big(\bigcup_{N_\theta,N_{\theta'}\in\mathcal{N}^\emptyset(\delta,\iota,\xi)} \mathcal{C}^\nu(\theta\Rightarrow\theta',\delta,\iota,\xi)\Big)$$

$$\mathcal{U}^\nu_n(\delta,\iota,\xi) \triangleq \{u \in \mathcal{U}^\nu(\delta,\iota,\xi) \mid |u| = n\}.$$

In our proof of the main theorem, the set $\mathcal{U}^\nu_{s(n)}(\delta,\iota,\xi)$ plays a role corresponding to $A^{s(n)}$ in the word case explained above. Note that in the word case we calculated the limit of some upper bound of $p(n)$; similarly, in our proof, we only need an upper bound of $\#(\mathcal{U}^\nu_n(\delta,\iota,\xi))$, which is given as follows.

Lemma 1. (upper bound of $\#(\mathcal{U}^\nu_n(\delta,\iota,\xi))$). *For any $\delta,\iota,\xi \geq 2$, there exists some constant $\overline{\gamma}(\delta,\iota,\xi) > 1$ such that $\#(\mathcal{U}^\nu_n(\delta,\iota,\xi)) = \mathrm{O}(\overline{\gamma}(\delta,\iota,\xi)^n)$.*

Proof Sketch. Given an unambiguous, irreducible and aperiodic regular tree grammar, adding a new terminal of the form a_N and a new rule of the form $N \longrightarrow a_N$ for each non-terminal N does not change the unambiguity, irreducibility and aperiodicity. Let $\overline{\mathcal{G}}^{\emptyset}(\delta,\iota,\xi)$ be the grammar obtained by applying this transformation to $\mathcal{G}^{\emptyset}(\delta,\iota,\xi)$. We can regard a tree of $\overline{\mathcal{G}}^{\emptyset}(\delta,\iota,\xi)$ as a normalized context, with a_{N_θ} considered a hole with typing θ. Then, clearly we have

$$\#(\mathcal{U}^\nu_n(\delta,\iota,\xi)) \leq \#\Big(\cup_{N\in\mathcal{N}^\emptyset(\delta,\iota,\xi)}\mathcal{L}_n\left(\overline{\mathcal{G}}^{\emptyset}(\delta,\iota,\xi),N\right)\Big).$$

Thus the required result follows from Theorem 2. □

4.2 Decomposition

As explained in the beginning of this section, to prove the parameterized infinite monkey theorem for terms, we need to decompose a λ-term into sufficiently many subcontexts of the term. Thus, in this subsection, we will

define a decomposition function $\widehat{\Phi}_m$ (where m is a parameter) that decomposes a term t into (i) a (sufficiently long) sequence P of 0/1-subcontexts of t such that every component u of P satisfies $|u| \geq m$, and (ii) a "second-order" context E (defined later), which is a remainder of extracting P from t. Figure 1 illustrates how a term is decomposed by $\widehat{\Phi}_3$. Here, the symbols $[\![\,]\!]$ in the second-order context on the right-hand side represents the original position of each subcontext $(\lambda y.[\,])x$, $\lambda z.\lambda *.z$, and $(\lambda *.y)\lambda z.z$.

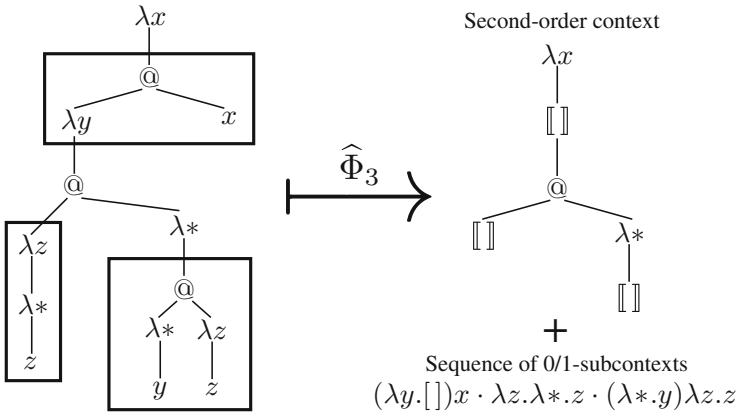


Fig. 1. Example of a decomposition

In order to define $\widehat{\Phi}_m$, let us give a precise definition of second-order contexts. The set of *second-order contexts*, ranged over by E, is defined by:

$$E ::= [\![\,]\!]_n^{\theta_1 \cdots \theta_k \Rightarrow \theta}[E_1] \cdots [E_k] \ (n \in \mathbb{N}) \mid x \mid \lambda \overline{x}^{\tau}.E \mid E_1 E_2.$$

Intuitively, the second-order context is an expression having holes of the form $[\![\,]\!]_n^{\kappa}$ (called *second-order holes*). In the second-order context $[\![\,]\!]_n^{\theta_1 \cdots \theta_k \Rightarrow \theta}[E_1] \cdots [E_k]$, $[\![\,]\!]_n^{\theta_1 \cdots \theta_k \Rightarrow \theta}$ should be filled with a $\theta_1 \cdots \theta_k \Rightarrow \theta$-context of size n, yielding a term whose typing is θ. We use the metavariable P for sequences of contexts. For a sequence of contexts $P = C_1 \cdot C_2 \cdots C_\ell$ and $i \leq \ell$, we write $\#(P)$ for the length ℓ, and $P.i$ for the i-th component C_i.

We define $|[\![\,]\!]_n^{\kappa}| \triangleq n$. We write $\mathtt{shn}(E)$ for the number of the second-order holes in E. For $i \leq \mathtt{shn}(E)$, we write $E.i$ for the i-th second-order hole (counted in the depth-first left-to-right pre-order). For a context C and a second-order hole $[\![\,]\!]_n^{\kappa}$, we write $C : [\![\,]\!]_n^{\kappa}$ if C is a κ-context of size n. For E and $P = C_1 \cdot C_2 \cdot \cdots \cdot C_{\mathtt{shn}(E)}$, we write $P : E$ if $C_i : E.i$ for each $i \leq \mathtt{shn}(E)$. We distinguish between second-order contexts with different annotations; for example, $[\![\,]\!]_0^{\langle\{x:\mathtt{o}\};\mathtt{o}\rangle \Rightarrow \langle\{x:\mathtt{o}\};\mathtt{o}\rangle}[x]$, $[\![\,]\!]_2^{\langle\{x:\mathtt{o}\};\mathtt{o}\rangle \Rightarrow \langle\{x:\mathtt{o}\};\mathtt{o}\rangle}[x]$ and $[\![\,]\!]_2^{\langle\{x:\mathtt{o}\to\mathtt{o}\};\mathtt{o}\to\mathtt{o}\rangle \Rightarrow \langle\{x:\mathtt{o}\to\mathtt{o}\};\mathtt{o}\to\mathtt{o}\rangle}[x]$ are different from each other. Note that every term can be regarded as a second-order context E such that $\mathtt{shn}(E) = 0$.

The bracket $[-]$ in a second-order context is just a syntactical representation rather than the substitution operation of contexts. Given E and C such that $\mathtt{shn}(E) \geq 1$ and $C : E\bullet 1$, we write $E[\![C]\!]$ for the second-order context obtained by replacing the leftmost second-order hole of E (i.e., $E\bullet 1$) with C (and by interpreting the syntactical bracket $[-]$ as the substitution operation). For example, we have: $((\lambda x.[\,][x][x])[\,])[\![\lambda y.y[\,][\,]]\!] = (\lambda x.(\lambda y.y[\,][\,])[x][x])[\,]\!] = (\lambda x.\lambda y.yxx)[\,]\!]$. Below we actually consider only second-order contexts whose second-order holes are of the form $[\,]^{\theta}_{n}$ or $[\,]^{\theta' \Rightarrow \theta}_{n}$.

We are now ready to define the decomposition function $\widehat{\Phi}_m$. We first prepare an auxiliary function $\Phi_m(t) = (E, u, P)$ such that (i) u is an auxiliary 0/1-subcontext, (ii) $E[\![u \cdot P]\!] = t$, and (iii) the size of each context in P is between m and $2m-1$. It is defined by induction on the size of t as follows:
If $|t| < m$, then $\Phi_m(t) \triangleq ([\,], t, \epsilon)$.
If $|t| \geq m$, then:

$$\Phi_m(\lambda\overline{x}^{\tau}.t_1) \triangleq \begin{cases} (E_1, \lambda\overline{x}^{\tau}.u_1, P_1) & \text{if } |\lambda\overline{x}^{\tau}.u_1| < m \\ ([\,][E_1], [\,], (\lambda\overline{x}^{\tau}.u_1) \cdot P_1) & \text{if } |\lambda\overline{x}^{\tau}.u_1| = m \end{cases}$$
$$\text{where } (E_1, u_1, P_1) = \Phi_m(t_1).$$

$$\Phi_m(t_1 t_2) \triangleq \begin{cases} ([\,][(E_1[\![u_1]\!])(E_2[\![u_2]\!])], [\,], P_1 \cdot P_2) & \text{if } |t_i| \geq m \ (i = 1, 2) \\ (E_1, u_1 t_2, P_1) & \text{if } |t_1| \geq m,\ |t_2| < m,\ |u_1 t_2| < m \\ ([\,][E_1], [\,], (u_1 t_2) \cdot P_1) & \text{if } |t_1| \geq m,\ |t_2| < m,\ |u_1 t_2| \geq m \\ (E_2, t_1 u_2, P_2) & \text{if } |t_1| < m,\ |t_1 u_2| < m \\ ([\,][E_2], [\,], (t_1 u_2) \cdot P_2) & \text{if } |t_1| < m,\ |t_1 u_2| \geq m \end{cases}$$
$$\text{where } (E_i, u_i, P_i) = \Phi_m(t_i) \quad (i = 1, 2).$$

Above, we have omitted the context-typing/size annotations of second-order holes for simplicity (see the full version [17] for details). The decomposition function $\widehat{\Phi}_m$ is then defined by $\widehat{\Phi}_m(t) \triangleq (E[\![u]\!], P)$ where $(E, u, P) = \Phi_m(t)$.

In the rest of this subsection, we show key properties of $\widehat{\Phi}_m$. We say that a 0/1-context u is *good for* m if u is either (i) a λ-abstraction where $|u| = m$; or (ii) an application $u_1 u_2$ where $|u_j| < m$ for each $j = 1, 2$. By the definition of $\widehat{\Phi}_m(t) = (E, P)$, every component u of P is good for m.

For $m \geq 2$, E and $1 \leq i \leq \mathtt{shn}(E)$, we define $\widehat{U}^m_{E\bullet i}(\delta, \iota, \xi)$, $\Lambda^m_E(\delta, \iota, \xi)$, and $\mathcal{B}^m_n(\delta, \iota, \xi)$ by:

$$\widehat{U}^m_{E\bullet i}(\delta, \iota, \xi) \triangleq \{u \in \mathcal{U}^{\nu}(\delta, \iota, \xi) \mid u : E\bullet i \text{ and } u \text{ is good for } m\}$$
$$\Lambda^m_E(\delta, \iota, \xi) \triangleq \{[E[\![u_1 \cdots u_{\mathtt{shn}(E)}]\!]]_{\alpha} \mid u_i \in \widehat{U}^m_{E\bullet i}(\delta, \iota, \xi) \text{ for } 1 \leq i \leq \mathtt{shn}(E)\}$$
$$\mathcal{B}^m_n(\delta, \iota, \xi) \triangleq \{E \mid (E, P) = \widehat{\Phi}_m(\nu([t]_{\alpha})) \text{ for some } [t]_{\alpha} \in \Lambda^{\alpha}_n(\delta, \iota, \xi)\}.$$

Intuitively, $\widehat{U}^m_{E\bullet i}(\delta, \iota, \xi)$ is the set of good contexts that can fill $E.i$, $\Lambda^m_E(\delta, \iota, \xi)$ is the set of terms obtained by filling the second-order holes of E with good contexts, and $\mathcal{B}^m_n(\delta, \iota, \xi)$ is the set of second-order contexts that can be obtained

by decomposing a term of size n. The following lemma states the key properties of $\widehat{\Phi}_m$.

Lemma 2. (decomposition). *Let $\delta, \iota, \xi \geq 0$ and $2 \leq m \leq n$.*

1. *$\Lambda_n^\alpha(\delta,\iota,\xi)$ is the disjoint union of $\Lambda_E^m(\delta,\iota,\xi)$'s, i.e., $\Lambda_n^\alpha(\delta,\iota,\xi) = \biguplus_{E\in\mathcal{B}_n^m(\delta,\iota,\xi)} \Lambda_E^m(\delta,\iota,\xi)$. Moreover, $\widehat{\Phi}_m(E[\![P]\!]) = (E,P)$ holds for any $P \in \prod_{1\leq i\leq \mathtt{shn}(E)} \widehat{U}_{E.i}^m(\delta,\iota,\xi)$.*
2. *$m \leq |E.i| < 2m$ $(1 \leq i \leq \mathtt{shn}(E))$ for every $E \in \mathcal{B}_n^m(\delta,\iota,\xi)$.*
3. *$\mathtt{shn}(E) \geq n/4m$ for every $E \in \mathcal{B}_n^m(\delta,\iota,\xi)$.*

The second and third properties say that $\widehat{\Phi}_m$ decomposes each term into sufficiently many contexts of appropriate size.

4.3 Explosive Context

Here, we show that each $\widehat{U}_{E.i}^m(\delta,\iota,\xi)$ contains at least one context that has a very long reduction sequence. To this end, we first prepare a special context $Expl_k^m$ that has a long reduction sequence, and shows that at least one element of $\widehat{U}_{E.i}^m(\delta,\iota,\xi)$ contains $Expl_k^m$ as a subcontext.

We define a "duplicating term" $Dup \triangleq \lambda x^{\mathtt{o}}.(\lambda x^{\mathtt{o}}.\lambda *^{\mathtt{o}}.x)xx$, and $Id \triangleq \lambda x^{\mathtt{o}}.x$. For two terms t, t' and integer $n \geq 1$, we define the "n-fold application" operation $\uparrow^n$ as $t \uparrow^0 t' \triangleq t'$ and $t \uparrow^n t' \triangleq t(t \uparrow^{n-1} t')$. For an integer $k \geq 2$, we define an order-k term

$$\overline{2}_k \triangleq \lambda f^{\tau(k)\to\tau(k)}.\lambda x^{\tau(k)}.f(fx)$$

where $\tau(i)$ is defined by $\tau(2) \triangleq \mathtt{o}$ and $\tau(i+1) \triangleq \tau(i)\to\tau(i)$.

Definition 5. (explosive context). Let $m \geq 1$ and $k \geq 2$ be integers and let

$$t \triangleq \nu\left(\lambda x^{\mathtt{o}}.\left((\overline{2}_k \uparrow^m \overline{2}_{k-1})\overline{2}_{k-2}\cdots\overline{2}_2\, Dup(Id\, x^\dagger)\right)\right)$$

where $x^\dagger$ is just variable x but we put $\dagger$ to refer to the occurrence. We define the *explosive context* $Expl_m^k$ (of m-fold and order k) as the 1-context obtained by replacing the "normalized" variable ${x_1}^\dagger$ in t with $[\,]$.

We state key properties of $Expl_m^k$ below. The proof of Item 3 is the same as that in [1]. The other items follow by straightforward calculation.

Lemma 3 (explosive).

1. *$\emptyset \vdash Expl_m^k[x_1] : \mathtt{o}\to\mathtt{o}$ is derivable.*
2. *$|Expl_m^k| = 8m + 8k - 3$.*
3. *$\mathtt{ord}(Expl_m^k[x_1]) = k$, $\mathtt{iar}(Expl_m^k[x_1]) = k$ and $\#(\mathbf{V}(Expl_m^k)) = 2$.*
4. *$Expl_m^k \in \mathcal{U}^\nu(\delta,\iota,\xi)$ if δ, $\iota \geq k$ and $\xi \geq 2$.*
5. *If a term t satisfies $Expl_m^k \preceq t$, then $\beta(t) \geq \mathbf{exp}_k(m)$ holds.*

We show that at least one element of $\widehat{U}_{E.i}^m(\delta,\iota,\xi)$ contains $Expl_{m'}^k$ as a subcontext.

Lemma 4 *Let $\delta, \iota, \xi \geq 2$ be integers and $k = \min\{\delta, \iota\}$. There exist integers $b, c \geq 2$ such that, for any $n \geq 1$, $m' \geq b$, $E \in \mathcal{B}_n^{cm'}(\delta, \iota, \xi)$ and $i \in \{1, \ldots, \mathtt{shn}(E)\}$, $\widehat{U}_{E.i}^{cm'}(\delta, \iota, \xi)$ contains u' such that $Expl_{m'}^k \preceq u'$.*

Proof Sketch. We pick $u'' \in \widehat{U}_{E.i}^{cm'}(\delta, \iota, \xi)$ and construct u' by replacing some subcontext u_0 of u'' with a 0/1-context of the form $S^\circ[Expl_{m'}^k[u^\circ]]$. Here S° and u° adjust the context typing and size of $Expl_{m'}^k$ and these can be obtained by using Proposition 2. The subcontext u_0 is chosen so that the goodness of u'' is preserved by this replacement. □

4.4 Proof Sketch of Theorem 1

We are now ready to prove the main theorem; see the full version [17] for details. For readability, we omit the parameters (δ, ι, ξ), and write $\Lambda_n^\alpha, \mathcal{U}_n^\nu, \Lambda_E^m, \widehat{U}_{E.i}^m$ and $\mathcal{B}_n^m$ for $\Lambda_n^\alpha(\delta, \iota, \xi), \mathcal{U}_n^\nu(\delta, \iota, \xi), \Lambda_E^m(\delta, \iota, \xi), \widehat{U}_{E.i}^m(\delta, \iota, \xi)$ and $\mathcal{B}_n^m(\delta, \iota, \xi)$ respectively.

Let $\overline{p}(n)$ be the probability that a randomly chosen normalized term t in Λ_n^α does *not* contain $Expl_{\lceil \log^{(2)}(n) \rceil}^k$ as a subcontext. By Item 3 of Lemma 3, it suffices to show $\lim_{n\to\infty} \overline{p}(n) = 0$. Let b and c be the constants in Lemma 4 and let $n \geq 2^{2^b}$, $m' = \lceil \log^{(2)}(n) \rceil$ and $m = cm'$. Then $m' \geq \log^{(2)}(n) \geq b$.

By Lemma 2, Λ_n^α can be represented as the disjoint union $\uplus_{E \in \mathcal{B}_n^m} \Lambda_E^m$. Let $\overline{\Lambda}_E^m$ be the subset of Λ_E^m that does not contain $Expl_{m'}^k$ as a subcontext. By Lemma 4, each of $\widehat{U}_{E.i}^m$ contains at least one element that has $Expl_{m'}^k$ as a subcontext. Furthermore, since $m \leq |E.i| < 2m$, we have $\#\left(\widehat{U}_{E.i}^m\right) \leq \#\left(\mathcal{U}_{2m+d}^\nu\right)$ for some constant d (see the full version [17]). Thus, we have

$$\frac{\#\left(\overline{\Lambda}_E^m\right)}{\#\left(\Lambda_E^m\right)} \leq \prod_{1 \leq i \leq \mathtt{shn}(E)} \left(1 - \frac{1}{\#\left(\widehat{U}_{E.i}^m\right)}\right) \leq \left(1 - \frac{1}{\#\left(\mathcal{U}_{2m+d}^\nu\right)}\right)^{\mathtt{shn}(E)}$$
$$\leq \left(1 - \frac{1}{\#\left(\mathcal{U}_{2m+d}^\nu\right)}\right)^{\frac{n}{4m}} \quad (\because \text{Item 3 of Lemma 2}).$$

Let $q(n)$ be the rightmost expression. Then we have

$$\overline{p}(n) = \frac{\sum_{E \in \mathcal{B}_n^m} \#\left(\overline{\Lambda}_E^m\right)}{\sum_{E \in \mathcal{B}_n^m} \#\left(\Lambda_E^m\right)} \leq \frac{\sum_{E \in \mathcal{B}_n^m} \left(q(n) \#\left(\Lambda_E^m\right)\right)}{\sum_{E \in \mathcal{B}_n^m} \#\left(\Lambda_E^m\right)}$$
$$= \frac{q(n) \sum_{E \in \mathcal{B}_n^m} \#\left(\Lambda_E^m\right)}{\sum_{E \in \mathcal{B}_n^m} \#\left(\Lambda_E^m\right)} = q(n) \leq \left(1 - \frac{1}{c' \overline{\gamma}(\delta, \iota, \xi)^{2m}}\right)^{\frac{n}{4m}} \quad (\because \text{Lemma 1})$$

for sufficiently large n. Finally, we can conclude that

$$\overline{p}(n) \leq \left(1 - \frac{1}{c' \overline{\gamma}(\delta, \iota, \xi)^{2c\lceil \log^{(2)}(n) \rceil}}\right)^{\frac{n}{4c\lceil \log^{(2)}(n) \rceil}} \longrightarrow 0 \quad (\text{as } n \longrightarrow \infty)$$

(see the full version [17] for the last convergence) as required. □

5 Related Work

As mentioned in Sect. 1, there are several pieces of work on probabilistic properties of untyped λ-terms [2–4]. David et al. [2] have shown that almost all untyped λ-terms are strongly normalizing, whereas the result is opposite for terms expressed in SK combinators.

Their former result implies that untyped λ-terms do *not* satisfy the infinite monkey theorem, i.e., for any term t, the probability that a randomly chosen term of size n contains t as a subterm tends to *zero*. Bendkowski et al. [4] proved that almost all terms in de Brujin representation are not strongly normalizing, by regarding the size of an index i is $i + 1$, instead of the constant 1. The discrepancies among those results suggest that this kind of probabilistic property is quite fragile and depends on the definition of the syntax and the size of terms. Thus, the setting of our paper, especially the assumption on the boundedness of internal arities and the number of variables is a matter of debate, and it would be interesting to study how the result changes for different assumptions.

We are not aware of similar studies on *typed* λ-terms. In fact, in their paper about combinatorial aspects of λ-terms, Grygiel and Lescanne [3] pointed out that the combinatorial study of typed λ-terms is difficult, due to the lack of (simple) recursive definition of typed terms. In the present paper, we have avoided the difficulty by making the assumption on the boundedness of internal arities and the number of variables (which is, as mentioned above, subject to a debate though).

In a larger context, our work may be viewed as an instance of the studies of average-case complexity ([20], Chap. 10), which discusses "typical-case feasibility". We are not aware of much work on the average-case complexity of problems with hyper-exponential complexity.

6 Conclusion

We have shown that almost every simply-typed λ-term of order k has a β-reduction sequence as long as $(k-2)$-fold exponential in the term size, under a certain assumption. To our knowledge, this is the first result of this kind for typed λ-terms. A lot of questions are left for future work, such as (i) whether our assumption (on the boundness of arities and the number of variables) is reasonable, and how the result changes for different assumptions, (ii) whether our result is optimal (e.g., whether almost every term has a k-fold exponentially long reduction sequence), and (iii) whether similar results hold for Terui's decision problems [14] and/or the higher-order model checking problem [6].

Acknowledgment. We would like to thank anonymous referees for useful comments. This work was supported by JSPS KAKENHI Grant Number JP15H05706.

References

1. Beckmann, A.: Exact bounds for lengths of reductions in typed lambda-calculus. J. Symb. Logic **66**(3), 1277–1285 (2001)
2. David, R., Grygiel, K., Kozic, J., Raffalli, C., Theyssier, G., Zaionc, M.: Asymptotically almost all λ-terms are strongly normalizing. Logical Method Comput. Sci. 9(2) (2013)
3. Grygiel, K., Lescanne, P.: Counting and generating lambda terms. J. Funct. Program. **23**(05), 594–628 (2013)
4. Bendkowski, M., Grygiel, K., Lescanne, P., Zaionc, M.: A natural counting of lambda terms. In: Freivalds, R.M., Engels, G., Catania, B. (eds.) SOFSEM 2016. LNCS, vol. 9587, pp. 183–194. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49192-8_15
5. Knapik, T., Niwiński, D., Urzyczyn, P.: Higher-order pushdown trees are easy. In: Nielsen, M., Engberg, U. (eds.) FoSSaCS 2002. LNCS, vol. 2303, pp. 205–222. Springer, Heidelberg (2002). doi:10.1007/3-540-45931-6_15
6. Ong, C.H.L.: On model-checking trees generated by higher-order recursion schemes. In: LICS 2006, pp. 81–90. IEEE Computer Society Press (2006)
7. Kobayashi, N.: Model-checking higher-order functions. In: Proceedings of PPDP 2009, pp. 25–36. ACM Press (2009)
8. Broadbent, C.H., Kobayashi, N.: Saturation-based model checking of higher-order recursion schemes. In: Proceedings of CSL 2013, vol. 23, pp. 129–148. LIPIcs (2013)
9. Ramsay, S., Neatherway, R., Ong, C.H.L.: An abstraction refinement approach to higher-order model checking. In: Proceedings of POpPL 2014 (2014)
10. Kobayashi, N.: Types and higher-order recursion schemes for verification of higher-order programs. ACM SIGPLAN Not. **44**, 416–428 (2009). ACM Press
11. Kobayashi, N., Sato, R., Unno, H.: Predicate abstraction and CEGAR for higher-order model checking. ACM SIGPLAN Not. **46**, 222–233 (2011). ACM Press
12. Ong, C.H.L., Ramsay, S.: Verifying higher-order programs with pattern-matching algebraic data types. ACM SIGPLAN Not. **46**, 587–598 (2011). ACM Press
13. Sato, R., Unno, H., Kobayashi, N.: Towards a scalable software model checker for higher-order programs. In: Proceedings of PEpPM 2013, pp. 53–62. ACM Press (2013)
14. Terui, K.: Semantic evaluation, intersection types and complexity of simply typed lambda calculus. In: 23rd International Conference on Rewriting Techniques and Applications (RTA 2012), vol. 15, pp. 323–338. LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2012)
15. Mairson, H.G.: Deciding ML typability is complete for deterministic exponential time. In: POPL, pp. 382–401. ACM Press (1990)
16. Kfoury, A.J., Tiuryn, J., Urzyczyn, P.: ML typability is dexptime-complete. In: Arnold, A. (ed.) CAAP 1990. LNCS, vol. 431, pp. 206–220. Springer, Heidelberg (1990). doi:10.1007/3-540-52590-4_50
17. Sin'ya, R., Asada, K., Kobayashi, N., Tsukada, T.: Almost every simply typed λ-term has a long β-reduction sequence (full version). http://www-kb.is.s.u-tokyo.ac.jp/ryoma/papers/fossacs17full.pdf
18. Heintze, N., McAllester, D.: Linear-time subtransitive control flow analysis. ACM SIGPLAN Not. **32**, 261–272 (1997)
19. Flajolet, P., Sedgewick, R.: Analytic Combinatorics, 1st edn. Cambridge University Press, New York (2009)
20. Goldreich, O.: Computational Complexity: A Conceptual Perspective. Cambridge University Press, Cambridge (2008)

Algebra and Coalgebra

Algebra, Coalgebra, and Minimization in Polynomial Differential Equations

Michele Boreale(✉)

Dipartimento di Statistica, Informatica, Applicazioni (DiSIA) "G. Parenti", Università di Firenze, Viale Morgagni 65, 50134 Firenze, Italy
michele.boreale@unifi.it

Abstract. We consider reasoning and minimization in systems of polynomial ordinary differential equations (ODEs). The ring of multivariate polynomials is employed as a syntax for denoting system behaviours. We endow polynomials with a transition system structure based on the concept of *Lie derivative*, thus inducing a notion of $\mathcal{L}$*-bisimulation*. Two states (variables) are proven $\mathcal{L}$-bisimilar if and only if they correspond to the same solution in the ODEs system. We then characterize $\mathcal{L}$-bisimilarity algebraically, in terms of certain ideals in the polynomial ring that are invariant under Lie-derivation. This characterization allows us to develop a complete algorithm, based on building an ascending chain of ideals, for computing the largest $\mathcal{L}$-bisimulation containing all valid identities that are instances of a user-specified template. A specific largest $\mathcal{L}$-bisimulation can be used to build a reduced system of ODEs, equivalent to the original one, but *minimal* among all those obtainable by linear aggregation of the original equations.

Keywords: Ordinary Differential Equations · Bisimulation · Minimization · Gröbner bases

1 Introduction

The past few years have witnessed a surge of interest in computational models based on ordinary differential equations (ODEs), ranging from continuous-time Markov chains (e.g. [3]), to process description languages oriented to bio-chemical systems (e.g. [4,10,32]), to deterministic approximations of stochastic systems (e.g. [15,31]), and to hybrid systems (e.g. [22,27,29]).

From a computational point of view, our motivations to study ODEs arises from the following problems.

1. *Reasoning*: provide methods to automatically prove and discover identities involving the system variables.
2. *Reduction*: provide methods to automatically reduce, and possibly minimize, the number of variables and equations of a system, in such a way that the reduced system retains all the relevant information of the original one.

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 71–87, 2017.
DOI: 10.1007/978-3-662-54458-7_5

Reasoning may help an expert (a chemist, a biologist, an engineer) to prove or to disprove certain system properties, even before actually solving, simulating or realizing the system. Often, the identities of interest take the form of *conservation laws*. For instance, chemical reactions often enjoy a mass conservation law, stating that the sum of the concentrations of two or more chemical species, is a constant. More generally, one would like tools to automatically *discover* all laws of a given form. Pragmatically, before actually solving or simulating a given system, it can be critical being able to reduce the system to a size that can be handled by a solver or a simulator.

Our goal is showing that these issues can be dealt with by a mix of algebraic and coalgebraic techniques. We will consider *initial value* problems, specified by a system of ODEs of the form $\dot{x}_i = f_i(x_1, ..., x_N)$, for $i = 1, ..., N$, plus initial conditions. The functions f_is are called *drifts*; here we will focus on the case where the drifts are multivariate polynomials in the variables $x_1, .., x_N$. Practically, the majority of functions found in applications is in, or can be encoded into this format (possibly under restrictions on the initial conditions), including exponential, trigonometric, logarithmic and rational functions.

A more detailed account of our work follows. We introduce the ring of multivariate polynomials as a syntax for denoting the *behaviours* induced by the given initial value problem (Sect. 2). In other words, a behaviour is any polynomial combination of the individual components $x_i(t)$ $(i = 1, .., N)$ of the (unique) system solution. We then endow the polynomial ring with a transition system, based on a purely syntactic notion of *Lie derivative* (Sect. 3). This structure naturally induces a notion of bisimulation over polynomials, $\mathcal{L}$*-bisimilarity*, that is in agreement with the underlying ODE' s. In particular, any two variables x_i and x_j are $\mathcal{L}$-bisimilar if and only the corresponding solutions are the same, $x_i(t) = x_j(t)$ (this generalizes to polynomial behaviours as expected). This way, one can prove identities between two behaviours, for instance conservation laws, by exhibiting bisimulations containing the given pair (in [8] we show how to enhance the resulting proof technique by introducing a polynomial version of the *up to* technique of [26]). In order to turn this method into a fully automated proof procedure, we first characterize $\mathcal{L}$-bisimulation algebraically, in terms of certain *ideals* in the polynomial ring that are invariant under Lie-derivation (Sect. 4). This characterization leads to an algorithm that, given a user-specified template, returns the set of all its instances that are valid identities in the system (Sect. 5). One may use this algorithm, for instance, to discover all the conservation laws of the system involving terms up to a given degree. The algorithm implies building an ascending chain of ideals until stabilization, and relies on a few basic concepts from Algebraic Geometry, notably Gröbner bases [16]. The output of the algorithm is in turn essential to build a reduced system of ODEs, equivalent to the original one, but featuring a *minimal* number of equations and variables, in the class of systems that can be obtained by linear aggregation from the original one (Sect. 6). We then illustrate the results of some simple experiments we have conducted using a prototype implementation (in Python) of our algorithms (Sect. 7). Our approach is mostly related to some recent work on equivalences for

ODEs by Cardelli et al. [11] and to work in the area of hybrid systems. We discuss this and other related work, as well as some possible directions for future work, in the concluding section (Sect. 8). Due to space limitations, most proofs and examples are omitted from the present version; they can be found in a technical report available online [8].

2 Preliminaries

Let us fix an integer $N \geq 1$ and a set of N distinct variables $x_1, ..., x_N$. We will denote by $\mathbf{x}$ the column[1] vector $(x_1, ..., x_N)^T$. We let $\mathbb{R}[\mathbf{x}]$ denote the set of multivariate polynomials in the variables $x_1, ..., x_N$ with coefficients in $\mathbb{R}$, and let p, q range over it. Here we regard polynomials as syntactic objects. Given an integer $d \geq 0$, by $\mathbb{R}_d[\mathbf{x}]$ we denote the set of polynomials of degree $\leq d$. As an example, $p = 2xy^2 + (1/5)wz + yz + 1$ is a polynomial of degree $\deg(p) = 3$, that is $p \in \mathbb{R}_3[x, y, z, w]$, with monomials xy^2, wz, yz and 1. Depending on the context, with a slight abuse of notation it may be convenient to let a polynomial denote the induced function $\mathbb{R}^N \to \mathbb{R}$, defined as expected. In particular, x_i can be seen as denoting the projection on the i-th coordinate.

A (polynomial) *vector field* is a vector of N polynomials, $F = (f_1, ..., f_N)^T$, seen as a function $F : \mathbb{R}^N \to \mathbb{R}^N$. A vector field F and an initial condition $\mathbf{x}_0 \in \mathbb{R}^N$ together define an *initial value problem* $\mathbf{\Phi} = (F, \mathbf{x}_0)$, often written in the following form

$$\mathbf{\Phi} : \begin{cases} \dot{\mathbf{x}}(t) = F(\mathbf{x}(t)) \\ \mathbf{x}(0) = \mathbf{x}_0. \end{cases} \tag{1}$$

The functions f_i in F are called *drifts* in this context. A *solution* to this problem is a differentiable function $\mathbf{x}(t) : D \to \mathbb{R}^N$, for some nonempty open interval $D \subseteq \mathbb{R}$ containing 0, which fulfills the above two equations, that is: $\frac{d}{dt}\mathbf{x}(t) = F(\mathbf{x}(t))$ for each $t \in D$ and $\mathbf{x}(0) = \mathbf{x}_0$. By the Picard-Lindelöf theorem [2], there exists a nonempty open interval D containing 0, over which there is a *unique* solution, say $\mathbf{x}(t) = (x_1(t), ..., x_N(t))^T$, to the problem. In our case, as F is infinitely often differentiable, the solution is seen to be *analytic* in D: each $x_i(t)$ admits a Taylor series expansion in a neighborhood of 0. For definiteness, we will take the domain of definition D of $\mathbf{x}(t)$ to be the largest symmetric open interval where the Taylor expansion from 0 of each of the $x_i(t)$ converges (possibly $D = \mathbb{R}$). The resulting vector function of t, $\mathbf{x}(t)$, is called the *time trajectory* of the system.

Given a differentiable function $g : E \to \mathbb{R}$, for some open set $E \subseteq \mathbb{R}^N$, the *Lie derivative of g along F* is the function $E \to \mathbb{R}$ defined as

$$\mathcal{L}_F(g) \triangleq \langle \nabla g, F \rangle = \sum_{i=1}^{N} \left(\frac{\partial g}{\partial x_i} \cdot f_i\right).$$

[1] *Vector* means column vector, unless otherwise specified.

The Lie derivative of the sum $h+g$ and product $h \cdot g$ functions obey the familiar rules

$$\mathcal{L}_F(h+g) = \mathcal{L}_F(h) + \mathcal{L}_F(g) \tag{2}$$
$$\mathcal{L}_F(h \cdot g) = h \cdot \mathcal{L}_F(g) + \mathcal{L}_F(h) \cdot g. \tag{3}$$

Note that $\mathcal{L}_F(x_i) = f_i$. Moreover if $p \in \mathbb{R}_d[\mathbf{x}]$ then $\mathcal{L}_F(p) \in \mathbb{R}_{d+d'}[\mathbf{x}]$, for some integer $d' \geq 0$ that depends on d and on F. This allows us to view the Lie derivative of polynomials along a polynomial field F as a purely syntactic mechanism, that is as a function $\mathcal{L}_F : \mathbb{R}[\mathbf{x}] \to \mathbb{R}[\mathbf{x}]$ that does not assume anything about the solution of (1). Informally, we can view p as a program, and taking Lie derivative of p can be interpreted as unfolding the definitions of the variables x_i's, according to the equations in (1) and to the formal rules for product and sum derivation, (2) and (3). We will pursue this view systematically in Sect. 3.

Example 1. Consider $N = 4$, $\mathbf{x} = (x, y, z, w)^T$ and the set of polynomials $\mathbb{R}[\mathbf{x}]$. The vector field $F = (xz + z, yw + z, z, w)^T$ and the initial condition $\mathbf{x}_0 = (0, 0, 1, 1)^T$ together define an initial value problem (with no particular physical meaning) $\mathbf{\Phi} = (F, \mathbf{x}_0)$. This problem can be equivalently written in the form

$$\begin{cases} \dot{x}(t) = x(t)z(t) + z(t) \\ \dot{y}(t) = y(t)w(t) + z(t) \\ \dot{z}(t) = z(t) \\ \dot{w}(t) = w(t) \\ \mathbf{x}(0) = \mathbf{x}_0 = (0, 0, 1, 1)^T. \end{cases} \tag{4}$$

As an example of Lie derivative, if $p = 2xy^2 + wz$, we have $\mathcal{L}_F(p) = 4wxy^2 + 2wz + 2xy^2z + 4xyz + 2y^2z$.

The connection between time trajectories, polynomials and Lie derivatives can be summed up as follows. For any polynomial $p \in \mathbb{R}[\mathbf{x}]$, the function $p(\mathbf{x}(t)) : D \to \mathbb{R}$, obtained by composing p as a function with the time trajectory $\mathbf{x}(t)$, is analytic: we let $p(t)$ denote the extension of this function over the largest symmetric open interval of convergence (possibly coinciding with $\mathbb{R}$) of its Taylor expansion from 0. We will call $p(t)$ the *polynomial behaviour induced by* p *and by* the initial value problem (1). The connection between Lie derivatives of p along F and the initial value problem (1) is given by the following equations, which can be readily checked. Here and in the sequel, we let $p(\mathbf{x}_0)$ denote the real number obtained by evaluating p at $\mathbf{x}_0$.

$$p(t)_{|t=0} = p(\mathbf{x}_0) \tag{5}$$
$$\frac{d}{dt}p(t) = (\mathcal{L}_F(p))(t). \tag{6}$$

More generally, defining inductively $\mathcal{L}_F^{(0)}(p) \stackrel{\triangle}{=} p$ and $\mathcal{L}_F^{(j+1)}(p) \stackrel{\triangle}{=} \mathcal{L}_F(\mathcal{L}_F^j(p))$, we have the following equation for the j-th derivative of $p(t)$ $(j = 0, 1, ...)$

$$\frac{d^j}{dt^j}p(t) = (\mathcal{L}_F^{(j)}(p))(t). \tag{7}$$

In the sequel, we shall often abbreviate $\mathcal{L}_F^{(j)}(p)$ as $p^{(j)}$, and shall omit the subscript $_F$ from $\mathcal{L}_F$ when clear from the context.

3 Coalgebraic Semantics of Polynomial ODEs

In this section we show how to endow the polynomial ring with a transition relation structure, hence giving rise to coalgebra. Bisimilarity in this coalgebra will correspond to equality between polynomial behaviours.

We recall that a *(Moore) coalgebra* (see e.g. [25]) with outputs in a set O is a triple $C = (S, \delta, o)$ where S is a set of *states*, $\delta : S \to S$ is a *transition* function, and $o : S \to O$ is an *output* function. A *bisimulation* in C is a binary relation $R \subseteq S \times S$ such that whenever $s\,R\,t$ then: (a) $o(s) = o(t)$, and (b) $\delta(s)\,R\,\delta(t)$. It is an (easy) consequence of the general theory of bisimulation that a largest bisimulation over S, called bisimilarity and denoted by $\sim$, exists, is the union of all bisimulation relations, and is an equivalence relation over S.

Given an initial value problem $\mathbf{\Phi} = (F, \mathbf{x}_0)$ of the form (1), the triple

$$C_{\mathbf{\Phi}} \triangleq (\mathbb{R}[\mathbf{x}], \mathcal{L}_F, o)$$

forms a coalgebra with outputs in $\mathbb{R}$, where: (1) $\mathbb{R}[\mathbf{x}]$ is the set of states; (2) $\mathcal{L}_F$ acts as the transition function; and (3) o defined as $o(p) \triangleq p(\mathbf{x}_0)$ is the output function. Note that this definition of coalgebra is merely syntactic, and does not presuppose anything about the solution of the given initial value problem. When the standard definition of bisimulation over coalgebras is instantiated to $C_{\mathbf{\Phi}}$, it yields the following.

Definition 1 ($\mathcal{L}$-bisimulation $\sim_{\mathbf{\Phi}}$). *Let $\mathbf{\Phi}$ be an initial value problem. A binary relation $R \subseteq \mathbb{R}[\mathbf{x}] \times \mathbb{R}[\mathbf{x}]$ is a* $\mathcal{L}$-bisimulation *if, whenever $p\,R\,q$ then: (a) $p(\mathbf{x}_0) = q(\mathbf{x}_0)$, and (b) $\mathcal{L}(p)\,R\,\mathcal{L}(q)$. The largest $\mathcal{L}$-bisimulation over $\mathbb{R}[\mathbf{x}]$ is denoted by $\sim_{\mathbf{\Phi}}$.*

We now introduce a new coalgebra with outputs in $\mathbb{R}$. Let $\mathcal{A}$ denote the family of real valued functions f such that f is analytic at 0 and f's domain of definition coincides with the open interval of convergence of its Taylor series (nonempty, centered at 0, possibly coinciding with $\mathbb{R}$)[2]. We define the coalgebra of analytic functions as

$$C_{\text{an}} \triangleq (\mathcal{A}, (\cdot)', o_{\text{an}})$$

where $(f)' = \frac{df}{dt}$ is the standard derivative, and $o_{\text{an}}(f) \triangleq f(0)$ is the output function. We recall that a *morphism* μ between two coalgebras with outputs in the same set, $\mu : C_1 \to C_2$, is a function from states to states that preserves

[2] Equivalently, $\mathcal{A}$ is the set of power series $f(t) = \sum_{j \geq 0} a_j t^j$ with a positive radius of convergence.

transitions ($\mu(\delta_1(s)) = \delta_2(\mu(s))$) and outputs ($o_1(s) = o_2(\mu(s))$). It is a standard (and easy) result of coalgebra that a morphism maps bisimilar states into bisimilar states: $s \sim_1 s'$ implies $\mu(s) \sim_2 \mu(s')$.

The coalgebra C_{an} has a special status, in that, given any coalgebra C with outputs in $\mathbb{R}$, *if* there is a morphism from C to C_{an}, this morphism is guaranteed to be *unique*[3]. For our purposes, it is enough to focus on $C = C_{\mathbf{\Phi}}$. We define the function $\mu : \mathbb{R}[\mathbf{x}] \to \mathcal{A}$ as

$$\mu(p) \stackrel{\triangle}{=} p(t).$$

Theorem 1 (coinduction). *μ is the unique morphism from $C_{\mathbf{\Phi}}$ to C_{an}. Moreover, the following* coinduction *principle is valid: $p \sim_{\mathbf{\Phi}} q$ in $C_{\mathbf{\Phi}}$ if and only if $p(t) = q(t)$ in $\mathcal{A}$.*

Theorem 1 permits proving polynomial relations among components $x_i(t)$ of $\mathbf{x}(t)$, say that $p(t) = q(t)$, by coinduction, that is, by exhibiting a suitable $\mathcal{L}$-bisimulation relating the polynomials p and q.

Example 2. For $N = 2$, consider the vector field $F = (x_2, -x_1)^T$ with the initial value $\mathbf{x}_0 = (0, 1)^T$. The binary relation $R \subseteq \mathbb{R}[x_1, x_2] \times \mathbb{R}[x_1, x_2]$ defined thus

$$R = \{(0, 0),\ (x_1^2 + x_2^2, 1)\}$$

is easily checked to be an $\mathcal{L}$-bisimulation. Thus we have proved the polynomial relation $x_1^2(t) + x_2^2(t) = 1$. Note that the unique solution to the given initial value problem is the pair of functions $\mathbf{x}(t) = (\sin(t), \cos(t))^T$. This way we have proven the familiar trigonometric identity $\sin(t)^2 + \cos(t)^2 = 1$.

This proof method can be greatly enhanced by a so called $\mathcal{L}$-*bisimulation up to* technique, in the spirit of [26]. See [8].

4 Algebraic Characterization of $\mathcal{L}$-bisimilarity

We first review the notion of polynomial ideal from Algebraic Geometry, referring the reader to e.g. [16] for a comprehensive treatment. A set of polynomials $I \subseteq \mathbb{R}[\mathbf{x}]$ is an *ideal* if: (1) $0 \in I$, (2) I is closed under sum $+$, (3) I is absorbing under product $\cdot$, that is $p \in I$ implies $h \cdot p \in I$ for each $h \in \mathbb{R}[\mathbf{x}]$. Given a set of polynomials S, the ideal generated by S, denoted by $\langle\ S\ \rangle$, is defined as

$$\left\{ \sum_{j=1}^{m} h_j p_j \ :\ m \geq 0, h_j \in \mathbb{R}[\mathbf{x}] \text{ and } p_j \in S,\ \text{for } j = 1, ..., m \right\}. \tag{8}$$

The polynomial coefficients h_j in the above definition are called *multipliers*. It can be proven that $\langle\ S\ \rangle$ is the smallest ideal containing S, which implies that $\langle\ \langle\ S\ \rangle\ \rangle = \langle\ S\ \rangle$. Any set S such that $\langle\ S\ \rangle = I$ is called a *basis* of I. Every

[3] Existence of a morphism is not guaranteed, though. In this sense, C_{an} is not *final*.

ideal in the polynomial ring $\mathbb{R}[\mathbf{x}]$ is finitely generated, that is has a finite basis (an instance of Hilbert's basis theorem).

$\mathcal{L}$-bisimulations can be connected to certain types of ideals. This connection relies on Lie derivatives. First, we define the Lie derivative of any set $S \subseteq \mathbb{R}[\mathbf{x}]$ as follows

$$\mathcal{L}(S) \triangleq \{\mathcal{L}(p) : p \in S\}.$$

We say that S is a *pre-fixpoint* of $\mathcal{L}$ if $\mathcal{L}(S) \subseteq S$. $\mathcal{L}$-bisimulations can be characterized as particular pre-fixpoints of $\mathcal{L}$ that are also ideals, called *invariants*.

Definition 2 (invariant ideals). *Let $\mathbf{\Phi} = (F, \mathbf{x}_0)$. An ideal I is a* $\mathbf{\Phi}$-invariant *if: (a) $p(\mathbf{x}_0) = 0$ for each $p \in I$, and (b) I is a pre-fixpoint of $\mathcal{L}_F$.*

We will drop the $\mathbf{\Phi}$- from $\mathbf{\Phi}$-*invariant* whenever this is clear from the context. The following definition and lemma provide the link between invariants and $\mathcal{L}$-bisimulation.

Definition 3 (kernel). *The* kernel *of a binary relation R is* $\ker(R) \triangleq \{p - q : p\, R\, q\}$.

Lemma 1. *Let R be a binary relation. If R is an $\mathcal{L}$-bisimulation then $\langle\ \ker(R)\ \rangle$ is an invariant. Conversely, given an invariant I, then $R = \{(p, q) : p - q \in I\}$ is an $\mathcal{L}$-bisimulation.*

Consequently, proving that $p \sim_{\mathbf{\Phi}} q$ is equivalent to exhibiting an invariant I such that $p - q \in I$. A more general problem than equivalence checking is finding *all* valid polynomial equations of a given form. We will illustrate an algorithm to this purpose in the next section.

The following result sums up the different characterization of $\mathcal{L}$-bisimilarity $\sim_{\mathbf{\Phi}}$. In what follows, we will denote the constant zero function in $\mathcal{A}$ simply by 0 and consider the following set of polynomials.

$$\mathcal{Z}_{\mathbf{\Phi}} \triangleq \{p : p(t) \text{ is identically } 0\}.$$

The following result also proves that $\mathcal{Z}_{\mathbf{\Phi}}$ is the largest $\mathbf{\Phi}$-invariant.

Theorem 2 ($\mathcal{L}$-bisimilarity via ideals). *For any pair of polynomials p and q: $p \sim_{\mathbf{\Phi}} q$ iff $p - q \in \ker(\sim_{\mathbf{\Phi}}) = \mathcal{Z}_{\mathbf{\Phi}} \overset{(1)}{=} \{p : p^{(j)}(\mathbf{x}_0) = 0$ for each $j \geq 0\} = \bigcup\{I : I$ is a $\mathbf{\Phi} -$ invariant$\}$.*

5 Computing Invariants

By Theorem 2, proving $p \sim_{\mathbf{\Phi}} q$ means finding an invariant I such that $p - q \in I \subseteq \mathcal{Z}_{\mathbf{\Phi}}$. More generally, we focus here on the problem of finding invariants that include a user-specified set of polynomials. In the sequel, we will make use of the following two basic facts about ideals, for whose proof we refer the reader to [16].

1. Any infinite ascending chain of ideals in a polynomial ring, $I_0 \subseteq I_1 \subseteq \cdots$, stabilizes at some finite k. That is, there is $k \geq 0$ such that $I_k = I_{k+j}$ for each $j \geq 0$.
2. The *ideal membership problem*, that is, deciding whether $p \in I$, given p and a finite set of S of *generators* (such that $I = \langle\, S \,\rangle$), is decidable (provided the coefficients used in p and in S can be finitely represented). The ideal membership will be further discussed later on in the section.

The main idea is introduced by the naive algorithm presented below.

A Naive Algorithm. Suppose we want to decide whether $p \in \mathcal{Z}_{\mathbf{\Phi}}$. It is quite easy to devise an algorithm that computes the smallest invariant containing p, or returns 'no' in case no such invariant exists, i.e. in case $p \notin \mathcal{Z}_{\mathbf{\Phi}}$. Consider the successive Lie derivatives of p, $p^{(j)} = \mathcal{L}^{(j)}(p)$ for $j = 0, 1, \ldots$. For each $j \geq 0$, let $I_j \triangleq \langle\, \{p^{(0)}, \ldots, p^{(j)}\} \,\rangle$. Let m be the least integer such that either

$$\text{(a) } p^{(m)}(\mathbf{x}_0) \neq 0, \qquad \text{or} \qquad \text{(b) } I_m = I_{m+1}.$$

If (a) occurs, then $p \notin \mathcal{Z}_{\mathbf{\Phi}}$, so we return 'no' (Theorem 2(1); if (b) occurs, then I_m is the least invariant containing p. Note that the integer m is well defined: $I_0 \subseteq I_1 \subseteq I_2 \subseteq \cdots$ forms an infinite ascending chain of ideals, which must stabilize in a finite number of steps (fact 1 at the beginning of the section).

Checking condition (b) amounts to deciding if $p^{(m+1)} \in I_m$. This is an instance of the ideal membership problem, which can be solved effectively. Generally speaking, given a polynomial p and finite set of polynomials S, deciding the ideal membership $p \in I = \langle\, S \,\rangle$ can be accomplished by first transforming S into a *Gröbner basis* G for I (via, e.g. the Buchberger's algorithm), then computing r, the *residual* of p modulo G (via a sort generalised division of p by G): one has that $p \in I$ if and only if $r = 0$ (again, this procedure can be carried out effectively only if the coefficients involved in p and S are finitely representable; in practice, one often confines to rational coefficients). We refer the reader to [16] for further details on the ideal membership problem. Known procedures to compute Gröbner bases have exponential worst-case time complexity, although may perform reasonably well in some concrete cases. One should in any case invoke such procedures parsimoniously.

Let us now introduce a more general version of the naive algorithm, which will be also able to deal with (infinite) *sets* of user-specified polynomials. First, we need to introduce the concept of template.

Templates. Polynomial templates have been introduced by Sankaranarayanan, Sipma and Manna in [27] as a means to compactly specify sets of polynomials. Fix a tuple of $n \geq 1$ of distinct *parameters*, say $\mathbf{a} = (a_1, \ldots, a_n)$, disjoint from $\mathbf{x}$. Let $Lin(\mathbf{a})$, ranged over by ℓ, be the set of *linear expressions* with coefficients in $\mathbb{R}$ and variables in $\mathbf{a}$; e.g. $\ell = 5a_1 + 42a_2 - 3a_3$ is one such expression[4].

[4] Differently from Sankaranarayanan et al. we do not allow linear expressions with a constant term, such as $2 + 5a_1 + 42a_2 - 3a_3$. This minor syntactic restriction does not practically affect the expressiveness of the resulting polynomial templates.

A *template* is a polynomial in $Lin(\mathbf{a})[\mathbf{x}]$, that is, a polynomial with linear expressions as coefficients; we let π range over templates. For example, the following is a template: $\pi = (5a_1 + (3/4)a_3)xy^2 + (7a_1 + (1/5)a_2)xz + (a_2 + 42a_3)$. Given a vector $v = (r_1, ..., r_n)^T \in \mathbb{R}^n$, we will let $\ell[v] \in \mathbb{R}$ denote the result of replacing each parameter a_i with r_i, and evaluating the resulting expression; we will let $\pi[v] \in \mathbb{R}[\mathbf{x}]$ denote the polynomial obtained by replacing each ℓ with $\ell[v]$ in π. Given a set $S \subseteq \mathbb{R}^n$, we let $\pi[S]$ denote the set $\{\pi[v] : v \in S\} \subseteq \mathbb{R}[\mathbf{x}]$.

The (formal) Lie derivative of π is defined as expected, once linear expressions are treated as constants; note that $\mathcal{L}(\pi)$ is still a template. It is easy to see that the following property is true: for each π and v, one has $\mathcal{L}(\pi[v]) = \mathcal{L}(\pi)[v]$. This property extends as expected to the j-th Lie derivative ($j \geq 0$)

$$\mathcal{L}^{(j)}(\pi[v]) = \mathcal{L}^{(j)}(\pi)[v]. \tag{9}$$

A Double Chain Algorithm. We present an algorithm that, given a template π with n parameters, returns a pair (V, J), where $V \subseteq \mathbb{R}^n$ is such that $\pi[\mathbb{R}^n] \cap \mathcal{Z}_{\mathbf{\Phi}} = \pi[V]$, and J is the smallest invariant that includes $\pi[V]$, possibly $J = \{0\}$. We first give a purely mathematical description of the algorithm, postponing its effective representation to the next subsection. The algorithm is based on building two chains of sets, a descending chain of vector spaces and an (eventually) ascending chain of ideals. The ideal chain is used to detect the stabilization of the sequence. For each $i \geq 0$, consider the sets

$$V_i \triangleq \{v \in \mathbb{R}^n : \pi^{(j)}[v](\mathbf{x}_0) = 0 \text{ for } j = 0, ..., i\} \tag{10}$$

$$J_i \triangleq \left\langle \bigcup_{j=1}^{i} \pi^{(j)}[V_i] \right\rangle. \tag{11}$$

It is easy to check that each $V_i \subseteq \mathbb{R}^n$ is a vector space over $\mathbb{R}$ of dimension $\leq n$. Now let $m \geq 0$ be the least integer such that the following conditions are *both* true:

$$V_{m+1} = V_m \tag{12}$$

$$J_{m+1} = J_m. \tag{13}$$

The algorithm returns (V_m, J_m). Note that the integer m is well defined: indeed, $V_0 \supseteq V_1 \supseteq \cdots$ forms an infinite descending chain of finite-dimensional vector spaces, which must stabilize in finitely many steps. In other words, we can consider the least m' such that $V_{m'} = V_{m'+k}$ for each $k \geq 1$. Then $J_{m'} \subseteq J_{m'+1} \subseteq \cdots$ forms an infinite ascending chain of ideals, which must stabilize at some $m \geq m'$. Therefore there must be some index m such that (12) and (13) are both satisfied, and we choose the least such m.

The next theorem states the correctness and relative completeness of this abstract algorithm. Informally, the algorithm will output the largest space V_m such that $\pi[V_m] \subseteq \mathcal{Z}_{\mathbf{\Phi}}$ and the smallest invariant J_m witnessing this inclusion. Note that, while typically the user will be interested in $\pi[V_m]$, J_m as well may

contain useful information, such as higher order, nonlinear conservation laws. We need a technical lemma.

Lemma 2. *Let V_m, J_m be the sets returned by the algorithm. Then for each $j \geq 1$, one has $V_m = V_{m+j}$ and $J_m = J_{m+j}$.*

Theorem 3 (correctness and relative completeness). *Let V_m, J_m be the sets returned by the algorithm for a polynomial template π.*

(a) $\pi[V_m] = \mathcal{Z}_{\mathbf{\Phi}} \cap \pi[\mathbb{R}^n]$;
(b) J_m is the smallest invariant containing $\pi[V_m]$.

Proof. Concerning part (a), we first note that $\pi[v] \in \mathcal{Z}_{\mathbf{\Phi}} \cap \pi[\mathbb{R}^n]$ means $(\pi[v])^{(j)}(\mathbf{x}_0) = \pi^{(j)}[v](\mathbf{x}_0) = 0$ for each $j \geq 0$ (Theorem 2(1)), which, by definition, implies $v \in V_j$ for each $j \geq 0$, hence $v \in V_m$. Conversely, if $v \in V_m = V_{m+1} = V_{m+2} = \cdots$ (here we are using Lemma 2), then by definition $(\pi[v])^{(j)}(\mathbf{x}_0) = \pi^{(j)}[v](\mathbf{x}_0) = 0$ for each $j \geq 0$, which implies that $\pi[v] \in \mathcal{Z}_{\mathbf{\Phi}}$ (again Theorem 2(1). Note that in proving both inclusions we have used property (9).

Concerning part (b), it is enough to prove that: (1) J_m is an invariant, (2) $J_m \supseteq \mathcal{Z}_{\mathbf{\Phi}} \cap \pi[\mathbb{R}^n]$, and (3) for any invariant I such that $\mathcal{Z}_{\mathbf{\Phi}} \cap \pi[\mathbb{R}^n] \subseteq I$, we have that $J_m \subseteq I$. We first prove (1), that J_m is an invariant. Indeed, for each $v \in V_m$ and each $j = 0, ..., m-1$, we have $\mathcal{L}(\pi^{(j)}[v]) = \pi^{(j+1)}[v] \in J_m$ by definition, while for $j = m$, since $v \in V_m = V_{m+1}$, we have $\mathcal{L}(\pi^{(m)}[v]) = \pi^{(m+1)}[v] \in J_{m+1} = J_m$ (note that in both cases we have used property (9)). Concerning (2), note that $J_m \supseteq \pi[V_m] = \mathcal{Z}_{\mathbf{\Phi}} \cap \pi[\mathbb{R}^n]$ by virtue of part (a). Concerning (3), consider any invariant $I \supseteq \mathcal{Z}_{\mathbf{\Phi}} \cap \pi[\mathbb{R}^n]$. We show by induction on $j = 0, 1, ...$ that for each $v \in V_m$, $\pi^{(j)}[v] \in I$; this will imply the wanted statement. Indeed, $\pi^{(0)}[v] = \pi[v] \in \mathcal{Z}_{\mathbf{\Phi}} \cap \pi[\mathbb{R}^n]$, as $\pi[V_m] \subseteq \mathcal{Z}_{\mathbf{\Phi}}$ by (a). Assuming now that $\pi^{(j)}[v] \in I$, by invariance of I we have $\pi^{(j+1)}[v] = \mathcal{L}(\pi^{(j)}[v]) \in I$ (again, we have used here property (9)).

According to Theorem 3(a), given a template π and $v \in \mathbb{R}^n$, checking if $\pi[v] \in \pi[V_m]$ is equivalent to checking if $v \in V_m$, which can be effectively done knowing a basis B_m of V_m. We show how to effectively compute such a basis in the following.

Effective Representation. For $i = 0, 1, ...$, we have to give effective ways to: (i) represent the sets V_i, J_i in (10) and (11); and, (ii) check the termination conditions (12) and (13). It is quite easy to address (i) and (ii) in the case of the vector spaces V_i. For each i, consider the linear expression $\pi^{(i)}(\mathbf{x}_0)$. By factoring out the parameters $a_1, ..., a_n$ in this expression, we can write, for a suitable (row) vector of coefficients $t_i = (t_{i1}, ..., t_{in}) \in \mathbb{R}^{1\times n}$: $\pi^{(i)}(\mathbf{x}_0) = t_{i1} \cdot a_1 + \cdots + t_{in} \cdot a_n$ The condition on $v \in \mathbb{R}^n$, $\pi^{(i)}[v](\mathbf{x}_0) = 0$, can then be translated into the linear constraint on v

$$t_i \cdot v = 0. \tag{14}$$

Letting $T_i \in \mathbb{R}^{i\times n}$ denote the matrix obtained by stacking the rows $t_1, ..., t_i$ on above the other, we see that V_i is the right null space of T_i. That is (here, $\mathbf{0}_i$ denotes the null vector in $\mathbb{R}^i$): $V_i = \{v \in \mathbb{R}^n \,:\, T_i v = \mathbf{0}_i\}$. Checking whether $V_i = V_{i+1}$ or not amounts then to checking whether the vector t_{i+1} is or not linearly dependent from the rows in T_i, which can be accomplished by standard and efficient linear algebraic techniques. In practice, the linear constraints (14) can be resolved and propagated via parameter elimination[5] incrementally, as soon as they are generated following computation of the derivatives $\pi^{(i)}$. Concerning the representation of the ideals J_i, we will use the following lemma[6].

Lemma 3. *Let $V \subseteq \mathbb{R}^n$ be a vector space with B as a basis, and $\pi_1, ..., \pi_k$ be templates. Then $\langle\ \cup_{j=1}^k \pi_j[V]\ \rangle = \langle\ \cup_{j=1}^k \pi_j[B]\ \rangle$.*

Now let B_i be a finite basis of V_i, which can be easily built from the matrix T_i. By the previous lemma, $\cup_{j=1}^i \pi^{(j)}[B_i]$ is a *finite* set of generators for J_i: this solves the representation problem. Concerning the termination condition, we note that, after checking that actually $V_i = V_{i+1}$, checking $J_i = J_{i+1}$ reduces to checking that $\pi^{(i+1)}[B_i] \subseteq \langle\ \cup_{j=1}^i \pi^{(j)}[B_i]\ \rangle = J_i \quad (*)$.

To check this inclusion, one can apply standard computer algebra techniques. For example, one can check if $\pi^{(i+1)}[b] \in J_i$ for each $b \in B_i$, thus solving $|B_i|$ ideal membership problems, for one and the same ideal J_i. As already discussed, this presupposes the computation of a Gröbner basis for J_i, a potentially expensive operation. One advantage of the above algorithm, over methods proposed in program analysis with termination conditions based on testing ideal membership (e.g. [24]), is that $(*)$ is not checked at every iteration, but only when $V_{i+1} = V_i$ (the latter a relatively inexpensive check).

Example 3. Consider the initial value problem of Example 1 and the template $\pi = a_1 x + a_2 y + a_3 z + a_4 w$. We run the double chain algorithm with this system and template as inputs. In what follows, $v = (v_1, v_2, v_3, v_4)^T$ will denote a generic vector in $\mathbb{R}^4$. Recall that $\mathbf{x} = (x, y, z, w)^T$ and $\mathbf{x}_0 = (0, 0, 1, 1)^T$.

- For each $v \in \mathbb{R}^4$: $\pi^{(0)}[v](\mathbf{x}_0) = (v_1 x + v_2 y + v_3 z + v_4 w)(\mathbf{x}_0) = 0$ if and only if $v \in V_0 \triangleq \{v \,:\, v_3 = -v_4\}$.
- For each $v \in V_0$: $\pi^{(1)}[v](\mathbf{x}_0) = (v_1 xz + v_1 z + v_2 wy + v_2 z + v_4 w - v_4 z)(\mathbf{x}_0) = 0$ if and only if $v \in V_1 \triangleq \{v \in V_0 : v_1 = -v_2\}$.
- For each $v \in V_1$: $\pi^{(2)}[v](\mathbf{x}_0) = (v_2 w^2 y + v_2 wy + v_2 wz - v_2 xz^2 - v_2 xz - v_2 z^2 + v_4 w - v_4 z)(\mathbf{x}_0) = 0$ if and only if $v \in V_2 \triangleq V_1$.
  Being $V_2 = V_1$, we also check if $J_2 = J_1$. A basis of V_1 is $B_1 = \{b_1, b_2\}$ with $b_1 = (-1, 1, 0, 0)^T$ and $b_2 = (0, 0, -1, 1)^T$. According to $(*)$, we have therefore to check if, for $\ell = 1, 2$: $\pi^{(2)}[b_\ell] \in J_1 \triangleq \langle\ \{\pi^{(0)}[b_1], \pi^{(0)}[b_2], \pi^{(1)}[b_1], \pi^{(1)}[b_2]\}\ \rangle$. With the help of a computer algebra system, one computes a Gröbner basis

[5] E.g., if for $\pi = a_1 x + a_2 y + a_3 x + a_4 w$ and $\mathbf{x}_0 = (0, 0, 1, 1)^T$, $\pi[v](\mathbf{x}_0) = 0$ is resolved by the substitution $[a_3 \mapsto -a_4]$.

[6] The restriction that linear expressions in templates do not contain constant terms is crucial here.

for J_1 as $G_1 = \{x - y, z - w\}$. Then one can reduce $\pi^{(2)}[b_1] = w^2y + wy + wz - xz^2 - xz - z^2$ modulo G_1 and obtain $\pi^{(2)}[b_1] = h_1(x - y) + h_2(z - w)$, with $h_1 = -z^2 - z$ and $h_2 = -wy - yz - y - z$, thus proving that $\pi^{(2)}[b_1] \in J_1$. One proves similarly that $\pi^{(2)}[b_2] \in J_1$. This shows that $J_2 = J_1$.

Hence the algorithm terminates with $m = 1$ and returns (V_1, J_1), or, concretely, (B_1, G_1). In particular, $x - y \in \mathcal{Z}_{\mathbf{\Phi}}$, or equivalently $x(t) = y(t)$. Similarly for $z - w$.

Remark 1 (linear systems). When the system of ODEs is linear, that is when the drifts f_i are linear functions of the x_i's, stabilization of the chain of vector spaces can be detected without resorting to ideals. The resulting single chain algorithm essentially boils down to the 'refinement' algorithm of [6, Theorem 2]. See [8] for details.

6 Minimization

We present a method for reducing the size of the an initial value problem. The basic idea is projecting the original system onto a suitably chosen subspace of $\mathbb{R}^N$. Consider the subspace $W \subseteq \mathbb{R}^N$ of all vectors that are orthogonal to $\mathbf{x}(t)$ for each t in the domain of definition D of the time trajectory, that is $W = \{v \in \mathbb{R}^N : \langle v\,, \mathbf{x}(t)\rangle = 0 \text{ for each } t \in D\}$. It is not difficult to prove (see [8]) that $W = V_m^{\perp}$, where V_m is the subspace returned by the double chain algorithm when fed with the linear template $\pi = \sum_{i=1}^{N} a_i x_i$. Let B the $N \times l$ ($l \leq m+1, N$) matrix whose columns are the vectors of an orthonormal basis of W. Then, for each t, $B^T\mathbf{x}(t)$ are the coordinates of $\mathbf{x}(t)$ in the subspace W w.r.t. the chosen basis. Consider now the following (reduced) initial value problem $\mathbf{\Psi}$, in the new variables $\mathbf{y} = (y_1, ..., y_l)^T$, obtained by projecting the original problem $\mathbf{\Phi}$ onto W. In [8], we prove the following result. In essence, all information about $\mathbf{\Phi}$ can be recovered exactly from the reduced $\mathbf{\Psi}$, which is the best possible linear reduction of $\mathbf{\Phi}$.

$$\mathbf{\Psi}: \begin{cases} \dot{\mathbf{y}}(t) = B^T F\,(B\mathbf{y}(t)) \\ \mathbf{y}(0) = B^T\mathbf{x}(0). \end{cases} \tag{15}$$

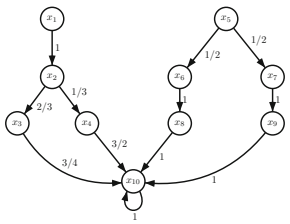
Theorem 4 (minimal exact reduction). *Let $\mathbf{y}(t)$ be the unique analytic solution of the (reduced) problem (15). Then, $\mathbf{x}(t) = B\mathbf{y}(t)$. Moreover, suppose for some $N \times k$ matrix C and vector function $\mathbf{z}(t)$, we have $\mathbf{x}(t) = C\mathbf{z}(t)$, for each $t \in D$. Then $k \geq l$.*

7 Examples

We have put a proof-of-concept implementation[7] of our algorithms at work on a few simple examples taken from the literature. We illustrate below two cases.

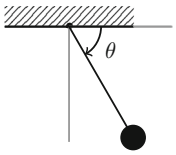
[7] Python code available at http://local.disia.unifi.it/boreale/papers/DoubleChain.py. Reported execution times relative to the pypy interpreter under Windows 8 on a core i5 machine.

Example 1: Linearity and Weighted Automata. The purpose of this example is to argue that, when the transition structure induced by the Lie-derivatives is considered, $\mathcal{L}$-bisimilarity is fundamentally a linear-time semantics. We first introduce *weighted automata*, a more compact[8] way of representing the transition structure of the coalgebra $C_{\mathbf{\Phi}}$.



A (finite or infinite) weighted automaton is like an ordinary automaton, save that both states and transitions are equipped with *weights* from $\mathbb{R}$. Given F and $\mathbf{x}_0$, we can build a weighted automaton with monomials as states, weighted transitions given by the rule $\alpha \xrightarrow{\lambda} \beta$ iff $\mathcal{L}_F(\alpha) = \lambda\beta + q$ for some polynomial q and real $\lambda \neq 0$, and where each state α is assigned weight $\alpha(\mathbf{x}_0)$. As an example, consider the weighted automaton on the right, where the state weights (not displayed) are 1 for x_{10}, and 0 for any other state. This automaton is generated (and in fact codes up) a system of ODEs with ten variables, where $\dot{x}_1 = x_2$, $\dot{x}_2 = (2/3)x_3 + (1/3)x_4$ etc., with the initial condition as specified by the state weights ($x_1(0) = 0$ etc.). The standard linear-time semantics of weighted automata (see [5,25] and references therein) is in full agreement with $\mathcal{L}$-bisimilarity [8]. As a consequence, in our example we have for instance that $x_1(t) = x_5(t)$. In fact, when invoked with this system and $\pi = \sum_{i=1}^{10} a_i x_i$ as inputs, the double chain algorithm terminates at $m = 2$ (in about 0.3 s; being this a linear system, Gröbner bases are never actually needed), returning $\pi[V_2] = (a_1(x_6 - x_7) + a_2(x_8 - x_9) + a_3(x_6 - x_2) + a_4(x_5 - x_1) + a_5(\frac{3}{2}x_8 - x_4) + a_6(\frac{3}{4}x_8 - x_3))[\mathbb{R}^6]$. This implies the expected $x_1 = x_5$, as well as other equivalences, and a 60% reduction in the minimal system.

Example 2: Nonlinear Conservation Laws. The law of the simple pendulum is $\frac{d^2}{dt^2}\theta = \frac{g}{\ell}\cos\theta$, where θ is the angle from the roof to the rod measured clockwise, ℓ is the length of the rod and g is gravity acceleration (see picture on the right). If we assume the initial condition $\theta(0) = 0$, this can be translated into the polynomial initial value problem below, where $\mathbf{x} = (\theta, \omega, x, y)^T$. The meaning of the variables is $\omega = \dot{\theta}$, $x = \cos\theta$ and $y = \sin\theta$. We assume for simplicity $\ell = 1$ and $g = 9$.



For this system, the double chain algorithm reports that there is no nontrivial linear conservation law (after $m = 6$ iterations and about 0.3 s). We then ask the algorithm to find all the conservation laws of order two, that is we use the template (α ranges over monomials) $\pi = \sum_{\alpha_i\,:\,\deg(\alpha_i)\leq 2} a_i\alpha_i$ as input. The algorithm terminates after $m = 16$ iterations (in about 7 s). The invariant J_{16} contains all the wanted conservation laws. The returned Gröbner basis for it is $G = \{x^2+y^2-1, \omega^2-18y\}$. The first term here just expresses the trigonometric

$$\begin{cases} \dot{\theta} = \omega \\ \dot{\omega} = \frac{g}{\ell}x \\ \dot{x} = -y\omega \\ \dot{y} = x\omega \\ \mathbf{x}(0) = (0,0,\ell,0)^T. \end{cases}$$

[8] At least for linear vector fields, the resulting weighted automaton is finite.

identity $(\cos\theta)^2 + (\sin\theta)^2 = 1$. Recalling that the (tangential) speed of the bob is $v = \ell\dot{\theta} = \ell\omega$, and that its vertical distance from the roof is $h = \ell\sin\theta = \ell y$, we see that the second term, considering our numerical values for ℓ, g, is equivalent to the equation $\frac{1}{2}v^2 = gh$, which, when multiplied by the mass m of the bob, yields the law of conservation of energy $\frac{1}{2}mv^2 = mgh$ (acquired kinetic energy = lost potential energy).

8 Future and Related Work

We briefly outline future work and related work below, referring the reader to [8] for a more comprehensive discussion.

Directions for future work. Scalability is an issue, as already for simple systems the Gröbner basis construction involved in the main algorithm can be computationally quite demanding. Further experimentation, relying on a well-engineered implementation of the method, and considering sizeable case studies, is called for in order to assess this aspect. Approximate reductions in the sense of System Theory [1] are also worth investigating.

Related work. Bisimulations for weighted automata are related to our approach, because, as argued in Sect. 7, Lie-derivation can be naturally represented by such an automaton. Algorithms for computing largest bisimulations on *finite* weighted automata have been studied by Boreale et al. [5,6]. A crucial ingredient in these algorithms is the representation of bisimulations as finite-dimensional vector spaces. Approximate version of this technique have also been recently considered in relation to Markov chains [7]. As discussed in Remark 1, in the case of linear systems, the algorithm in the present paper reduces to that of [5,6]. Algebraically, moving from linear to polynomial systems corresponds to moving from vector spaces to ideals, hence from linear bases to Gröbner bases. From the point of view automata, this step leads to considering infinite weighted automata. In this respect, the present work may be also be related to the automata-theoretic treatment of linear ODEs by Fliess and Reutenauer [17].

Although there exists a rich literature dealing with linear aggregation of systems of ODEs (e.g. [1,19,21,30]), we are not aware of fully automated approaches to minimization (Theorem 4) with the notable exception of a series of recent works by Cardelli and collaborators [11–13]. Mostly related to ours is [11]. There, for an extension of the polynomial ODE format called IDOL, the authors introduce two flavours of *differential equivalence*, called *Forward* (FDE) and *Backward* (BDE). They provide a symbolic, SMT-based partition refining algorithms to compute the largest equivalence of each type. While FDE is unrelated with our equivalence, BDE can be compared directly to our $\mathcal{L}$-bisimulation. An important difference is that BDE is stricter than necessary, as it may tell apart variables that have the same solution. This is not the case with $\mathcal{L}$-bisimilarity, which is, in this respect, correct and complete. An important consequence of this difference is that the quotient system produced by BDE is not minimal, whereas

that produced by $\mathcal{L}$-bisimulation is in a precise sense. For example, BDE finds no reductions at all in the linear system[9] of Sect. 7, whereas $\mathcal{L}$-bisimulation, as seen, leads to a 60% reduction. Finally, the approach of [11] and ours rely on two quite different algorithmic decision techniques, SMT and Gröbner bases, both of which have exponential worst-case complexity. As shown by the experiments reported in [11], in practice BDE and FDE have proven quite effective at system reduction. At the moment, we lack similar experimental evidence for $\mathcal{L}$-bisimilarity.

The seminal paper of Sankaranarayanan, Sipma and Manna [27] introduced polynomial ideals to find invariants of hybrid systems. Indeed, the study of the safety of hybrid systems can be shown to reduce constructively to the problem of generating invariants for their differential equations [23]. The results in [27] have been subsequently refined and simplified by Sankaranarayanan using *pseudoideals* [28], which enable the discovery of polynomial invariants of a special form. Other authors have adapted this approach to the case of imperative programs, see e.g. [9,20,24] and references therein. Reduction and minimization seem to be not a concern in this field.

Still in the field of formal verification of hybrid systems, mostly related to ours is Ghorbal and Platzer's recent work on polynomial invariants [18]. One one hand, they characterize algebraically invariant regions of vector fields – as opposed to initial value problems, as we do. On the other hand, they offer sufficient conditions under which the trajectory induced by a specific initial value satisfies all instances of a polynomial template (cf. [18, Proposition 3]). The latter result compares with ours, but the resulting method appears to be not (relatively) complete in the sense of our double chain algorithm. Moreover, the computational prerequisites of [18] (symbolic linear programming, exponential size matrices, symbolic root extraction) are very different from ours, and much more demanding. Again, minimization is not addressed.

Acknowledgments. The author has benefited from stimulating discussions with Mirco Tribastone.

References

1. Antoulas, A.C.: Approximation of Large-scale Dynamical Systems. SIAM, Philadelphia (2005)
2. Arnold, V.I.: Ordinary Differential Equations. The MIT Press, Cambridge (1978). ISBN 0-262-51018-9
3. Bernardo, M.: A survey of markovian behavioral equivalences. In: Bernardo, M., Hillston, J. (eds.) SFM 2007. LNCS, vol. 4486, pp. 180–219. Springer, Heidelberg (2007). doi:10.1007/978-3-540-72522-0_5
4. Blinov, M.L., Faeder, J.R., Goldstein, B., Hlavacek, W.S.: BioNet-Gen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. Bioinformatics **20**(17), 3289–3291 (2004)
5. Bonchi, F., Bonsangue, M.M., Boreale, M., Rutten, J.J.M.M., Silva, A.: A coalgebraic perspective on linear weighted automata. Inf. Comput. **211**, 77–105 (2012)

[9] Checked with the Erode tool by the same authors [14].

6. Boreale, M.: Weighted bisimulation in linear algebraic form. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 163–177. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04081-8_12
7. Boreale, M.: Analysis of probabilistic systems via generating functions and Padé approximation. In: Halldórsson, M.M., Iwama, K., Kobayashi, N., Speckmann, B. (eds.) ICALP 2015. LNCS, vol. 9135, pp. 82–94. Springer, Heidelberg (2015). doi:10.1007/978-3-662-47666-6_7. http://local.disia.unifi.it/wp_disia/2016/wp_disia_2016_10.pdf
8. Boreale, M.: Algebra, coalgebra, and minimization in polynomial differential equations. DiSIA working paper, January 2017. http://local.disia.unifi.it/wp_disia/2017wp_disia_2017_01.pdf
9. Cachera, D., Jensen, T., Jobin, A., Kirchner, F.: Inference of polynomial invariants for imperative programs: a farewell to Gröbner bases. In: Miné, A., Schmidt, D. (eds.) SAS 2012. LNCS, vol. 7460, pp. 58–74. Springer, Heidelberg (2012). doi:10.1007/978-3-642-33125-1_7
10. Cardelli, L.: On process rate semantics. Theor. Comput. Sci. **391**(3), 190–215 (2008)
11. Cardelli, L., Tribastone, M., Tschaikowski, M., Vandin, A.: Symbolic computation of differential equivalences. In: POPL (2016)
12. Cardelli, L., Tribastone, M., Tschaikowski, M., Vandin, A.: Efficient syntax-driven lumping of differential equations. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 93–111. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49674-9_6
13. Cardelli, L., Tribastone, M., Tschaikowski, M., Vandin, A., Networks, comparing chemical reaction : a categorical and algorithmic perspective. In: LICS (2016, to appear)
14. Cardelli, L., Tribastone, M., Tschaikowski, M., Vandin, A.: ERODE: evaluation and reduction of ordinary differential equations. http://sysma.imtlucca.it/tools/erode/
15. Ciocchetta, F., Hillston, J.: Bio-PEPA: a framework for the modelling and analysis of biological systems. Theor. Comput. Sci. **410**(33–34), 3065–3084 (2009)
16. Cox, D., Little, J., O'Shea, D.: Ideals, Varieties, and Algorithms An Introduction to Computational Algebraic Geometry and Commutative Algebra. Undergraduate Texts in Mathematics. Springer, New York (2007)
17. Fliess, M., Reutenauer, C.: Theorie de Picard-Vessiot des Systèmes Reguliers. Colloque Nat. CNRS-RCP567, Belle-ile sept. in Outils et Modèles Mathématiques pour l'Automatique l'Analyse des systèmes et le tratement du signal. CNRS, 1983 (1982)
18. Ghorbal, K., Platzer, A.: Characterizing algebraic invariants by differential radical invariants. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 279–294. Springer, Heidelberg (2014). doi:10.1007/978-3-642-54862-8_19. http://reports-archive.adm.cs.cmu.edu/anon/2013/CMU-CS-13-129.pdf
19. Li, G., Rabitz, H., Tóth, J.: A general analysis of exact nonlinear lumping in chemical kinetics. Chem. Eng. Sci. **49**(3), 343–361 (1994)
20. Müller-Olm, M., Seidl, H.: Computing polynomial program invariants. Inf. Process. Lett. **91**(5), 233–244 (2004)
21. Okino, M.S., Mavrovouniotis, M.L.: Simplification of mathematical models of chemical reaction systems. Chem. Rev. **2**(98), 391–408 (1998)
22. Platzer, A.: Differential dynamic logic for hybrid systems. J. Autom. Reason. **41**(2), 143–189 (2008)

23. Platzer, A.: Logics of dynamical systems. In: LICS, pp. 13–24. IEEE (2012)
24. Rodríguez-Carbonell, E., Kapur, D.: Generating all polynomial invariants in simple loops. J. Symb. Comput. **42**(4), 443–476 (2007)
25. Rutten, J.J.M.M.: Behavioural differential equations: a coinductive calculus of streams, automata, and power series. Theor. Comput. Sci. **308**(1–3), 1–53 (2003)
26. Sangiorgi, D.: Beyond bisimulation: the "up-to" techniques. In: Boer, F.S., Bonsangue, M.M., Graf, S., Roever, W.-P. (eds.) FMCO 2005. LNCS, vol. 4111, pp. 161–171. Springer, Heidelberg (2006). doi:10.1007/11804192_8
27. Sankaranarayanan, S., Sipma, H., Manna, Z.: Non-linear loop invariant generation using Gröbner bases. In: POPL (2004)
28. Sankaranarayanan, S.: Automatic invariant generation for hybrid systems using ideal fixed points. In: HSCC 2010, pp. 221–230 (2010)
29. Tiwari, A.: Approximate reachability for linear systems. In: Maler, O., Pnueli, A. (eds.) HSCC 2003. LNCS, vol. 2623, pp. 514–525. Springer, Heidelberg (2003). doi:10.1007/3-540-36580-X_37
30. Tóth, J., Li, G., Rabitz, H., Tomlin, A.S.: The effect of lumping and expanding on kinetic differential equations. SIAM J. Appl. Math. **57**(6), 1531–1556 (1997)
31. Tribastone, M., Gilmore, S., Hillston, J.: Scalable differential analysis of process algebra models. IEEE Trans. Softw. Eng. **38**(1), 205–219 (2012)
32. Voit, E.O.: Biochemical systems theory: a review. ISRN Biomath. **2013**, 53 (2013)

Equational Theories of Abnormal Termination Based on Kleene Algebra

Konstantinos Mamouras(✉)

University of Pennsylvania, Philadelphia, USA
mamouras@seas.upenn.edu

Abstract. We study at an abstract level imperative while programs with an explicit *fail* operation that causes abnormal termination or irreparable failure, and a *try-catch* operation for error handling. There are two meaningful ways to define the semantics of such programs, depending on whether the final state of the computation can be observed upon failure or not. These two semantics give rise to different equational theories. We investigate these two theories in the abstract framework of Kleene algebra, and we propose two simple and intuitive equational axiomatizations. We prove very general conservativity results, from which we also obtain decidability and deductive completeness of each of our calculi with respect to the intended semantics.

1 Introduction

The computations of imperative programs are typically divided into two distinct categories: those that terminate normally at some final state (thus possibly yielding an output), and those that do not terminate or, as we say, that diverge. However, for most realistic programs there is also the possibility of *failure*, which has to be distinguished from normal termination. When we say failure here, we are referring to the computational phenomenon where an executing program has to stop immediately because something "bad" has happened that prevents it from continuing with its computation. There are numerous examples of such behavior: a memory access error, a division by zero, the failure of a user-defined assertion, and so on. Depending on the context, this kind of irreparable failure is described with various names: abnormal or abrupt termination, (uncaught) exception, program crash, etc.

An important point to be made is that when failure is a possibility there are two different (both very meaningful) ways of defining semantics, depending on whether the final state of the computation can be observed upon failure or not. Let us consider a standard way of describing the intended input-output behavior of imperative programs by describing how they execute on an idealized machine. This is the so called *operational semantics*, and it amounts to giving a detailed description of the individual steps of the computation as it mutates the program state. In our setting, where failure is a possibility, a computation of the program f can take one of the following three forms:

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 88–105, 2017.
DOI: 10.1007/978-3-662-54458-7_6

$$\text{normal termination: } \langle x, f\rangle \to \langle x', f'\rangle \to \cdots \to \langle y, 1\rangle$$
$$\text{divergence: } \langle x, f\rangle \to \langle x', f'\rangle \to \cdots$$
$$\text{abnormal termination: } \langle x, f\rangle \to \langle x', f'\rangle \to \cdots \to \langle y, \mathsf{fail}(e)\rangle$$

The letters x, x', y above represent entire program states, the constant 1 represents the program that immediately terminates normally, and the constant $\mathsf{fail}(e)$ represents the program that immediately terminates abnormally with error code (exception) e. We summarize the input-output behavior of the program f by including the pair $x \mapsto \langle y, 1\rangle$ when the normally terminating computation $\langle x, f\rangle \to \cdots \to \langle y, 1\rangle$ is possible, and the pair $x \mapsto \langle y, \mathsf{fail}(e)\rangle$ when the abnormally terminating computation $\langle x, f\rangle \to \cdots \to \langle y, \mathsf{fail}(e)\rangle$ is possible. Thus, the *meaning* or *denotation* of a program f that computes by transforming a state space X is given by a function of type $X \to P(X \oplus (X \times E))$, where P denotes the powerset functor, E is the set of error codes (exceptions), and $\oplus$ is the coproduct (disjoint union) operation. Intuitively, using this semantics we are able to observe the final state of the computation upon abnormal termination. However, it may sometimes be appropriate to disregard the final state when the program fails. In this latter case, the meaning of the program is given by a function of type $X \to P(X \oplus E)$. Informally, our second semantics is derived from the first one by "forgetting" the final state in the case of failure. To sum this up:

$$\text{Final state } \textit{observable} \text{ upon failure:} \quad X \to P(X \oplus (X \times E))$$
$$\text{Final state } \textit{not observable} \text{ upon failure:} \quad X \to P(X \oplus E)$$

In both cases, the powerset functor P allows us to accomodate nondeterminism. That is, when a program starts at some state, there may be several possible computations and we want to record all their possible outcomes.

These two different ways of assigning meaning to programs that we described in the previous paragraph give rise to ***two distinct notions of equivalence***. Let us write $f \equiv g$ if the programs f and g have the same meaning under the first semantics involving functions $X \to P(X \oplus (X \times E))$. We write $\approxeq$ to denote the equivalence induced by the semantics involving functions $X \to P(X \oplus E)$. An immediate observation is that the equivalence $\equiv$ is *finer* than $\approxeq$, which means that $f \equiv g$ implies $f \approxeq g$. In fact, $\equiv$ is *strictly finer* than $\approxeq$, because:

$$x := 0; \mathsf{fail}(e) \not\equiv x := 1; \mathsf{fail}(e) \qquad x := 0; \mathsf{fail}(e) \approxeq x := 1; \mathsf{fail}(e)$$

where ; is the sequential composition operation. The programs $x := 0; \mathsf{fail}(e)$ and $x := 1; \mathsf{fail}(e)$ both terminate abnormally at states with $x = 0$ and $x = 1$ respectively. When the final states can be observed, the two programs can be differentiated by looking at the final value of the variable x. But, when the final states are unobservable, then the programs are semantically the same.

We are interested here in axiomatizing completely the two equational theories given by the equivalences $\equiv$ and $\approxeq$. Of course, this endeavor is not possible for typical interpreted programming languages that are Turing-complete. For such languages the equational theories are not recursively enumerable, and hence

they admit no complete effective axiomatization. We will therefore work in a very abstract uninterpreted setting, where the building blocks of our imperative programming language are abstract actions $a, b, c, \ldots$ with no fixed interpretation. This is the setting of Kleene algebra (KA) [12] and Kleene algebra with tests (KAT) [13], whose language includes the following program structuring operations: sequential composition $\cdot$, nondeterministic choice $+$, and nondeterministic iteration *. KAT involves a special syntactic category of *tests* $p, q, r, \ldots$, which can model the guards of conditionals and while loops:

$$\mathsf{if}\, p\, \mathsf{then}\, f\, \mathsf{else}\, q \triangleq p \cdot f + \neg p \cdot g \qquad \mathsf{while}\, p\, \mathsf{do}\, f \triangleq (p \cdot f)^* \cdot \neg p$$

It is therefore sufficient to consider the language of KAT for logical investigations that concern imperative nondeterministic programs with while loops.

If failure is not a possibility, then the axioms of KA and KAT are very well suited for abstract reasoning about programs. However, if failure is possible then the property $f \cdot 0 = 0$, which is an axiom of KA, is ***no longer valid***. The constant 0 denotes the program that always diverges, and we expect:

$$\mathsf{fail}(e) \cdot 0 = \mathsf{fail}(e) \qquad \mathsf{fail}(e) \cdot 0 \neq 0$$

This suggests that, in order to reason conveniently about programs that can fail, we need to enrich the syntax of KAT and capture the essential properties of the fail operation. As a first step, we introduce the simplest type discipline necessary by distinguishing the programs that contain no occurrence of fail. In algebraic jargon, this is a *two-sorted* approach: we consider a sub-sort of fail-*free* programs in addition to the sort of all programs. The basic algebraic theory that we propose, which we call **FailKAT**, differs from KAT by weakening the axiom $f \cdot 0 = 0$ and by introducing just one extra equation for $\mathsf{fail}(e)$:

$$f \cdot 0 = 0, \text{ where } f \text{ is a fail-free program}$$
$$\mathsf{fail}(e) \cdot f = \mathsf{fail}(e), \text{ where } f \text{ can be any program}$$

As we shall see, these modifications are exactly what is needed to reason about failure (for the $\equiv$ equivalence). They are absolutely necessary, and they introduce no additional syntactic complications than what is strictly required. Even more interestingly, we can axiomatize the coarser $\approxeq$ equivalence by adding to the **FailKAT** calculus just one more equational axiom (The partial order $\leq$ is defined in every KA in terms of nondeterministic choice as follows: $x \leq y$ iff $x + y = y$. See, for example, [12].):

$$f \cdot \mathsf{fail}(e) \leq \mathsf{fail}(e), \text{ where } f \text{ is a fail-free program.}$$

Let us call this extended calculus **FailTKAT**. The above property has a straightforward intuitive explanation. When we are disallowed to observe the final state upon failure, then any program that we execute before failing has no observable effect other than possibly causing divergence (hence we have $\leq$ instead of $=$).

$$
\begin{array}{lcl}
i := 0 & \equiv & i := 0 \\
\mathsf{while}\,(i < n)\,\mathsf{do} & & \mathsf{while}\,(i < n)\,\mathsf{do} \\
\quad \mathsf{assert}\ (0 \le i) \wedge (i < X.\mathsf{length}) & & \quad \mathsf{if}\,(i \ge X.\mathsf{length})\,\mathsf{then}\,\mathsf{fail} \\
\quad X[i] := 0 & & \quad X[i] := 0 \\
\quad i := i + 1 & & \quad i := i + 1
\end{array}
$$

$$(i := 0) \equiv (i := 0); (i \ge 0) \qquad (i \ge 0); (X[i] := 0) \equiv (X[i] := 0); (i \ge 0)$$
$$\neg(i < X.\mathsf{length}) \equiv (i \ge X.\mathsf{length}) \qquad (i \ge 0); (i := i + 1) \equiv (i \ge 0); (i := i + 1); (i \ge 0)$$

Fig. 1. The logical system **FailKAT** can be used for verifying program optimizations.

We further enrich the language with a *try-catch construct* for exception handling. The operational semantics that we consider is the standard one used in imperative programming languages:

$$\frac{\langle x, f\rangle \to \langle x', f'\rangle}{\langle x, \mathsf{try}\ f\ \mathsf{catch}(e)\ g\rangle \to \langle x', \mathsf{try}\ f'\ \mathsf{catch}(e)\ g\rangle} \qquad \begin{array}{c} \langle x, \mathsf{try}\,1\,\mathsf{catch}(e)\,g\rangle \to \langle x, 1\rangle \\ \langle x, \mathsf{try}\,\mathsf{fail}(e)\,\mathsf{catch}(e)\,g\rangle \to \langle x, g\rangle \end{array}$$
$$\langle x, \mathsf{try}\,\mathsf{fail}(d)\,\mathsf{catch}(e)\,g\rangle \to \langle x, \mathsf{fail}(d)\rangle, \text{ when } d \neq e$$

As we will prove, the following five equational axioms

$$
\begin{aligned}
\mathsf{try}\,1\,\mathsf{catch}(e)\,g &= 1 \\
\mathsf{try}\,\mathsf{fail}(d)\,\mathsf{catch}(e)\,g &= \mathsf{fail}(d), \text{ when } d \neq e \\
\mathsf{try}\,\mathsf{fail}(e)\,\mathsf{catch}(e)\,g &= g \\
\mathsf{try}\,(f + g)\,\mathsf{catch}(e)\,h &= (\mathsf{try}\,f\,\mathsf{catch}(e)\,h) + (\mathsf{try}\,g\,\mathsf{catch}(e)\,h) \\
\mathsf{try}\,(f \cdot g)\,\mathsf{catch}(e)\,h &= f \cdot (\mathsf{try}\,g\,\mathsf{catch}(e)\,h), \text{ when } f \text{ is fail-free}
\end{aligned}
$$

are sufficient to derive all the algebraic properties of the try-catch construct. We note that fail and try-catch can also be used to model several other useful non-local control flow constructs, such as break, continue and return.

The problem of axiomatizing the algebraic properties of the $\mathsf{try}\ \ldots\ \mathsf{catch}(e)\ \ldots$ and $\mathsf{fail}(e)$ constructs is more than of purely mathematical interest. There are several useful program optimizations that concern user-defined assertions and statements that can cause failure. **FailKAT** can offer a formal compositional approach to the verification of such program optimizations. For example, in Fig. 1 we see a valid program optimization that simplifies a user-defined assertion, where the statement $\mathsf{assert}\ p$ is syntactic sugar for $\mathsf{if}\,p\,\mathsf{then}\,1\,\mathsf{else}\,\mathsf{fail}$. The optimization is expressed as an equation, and it can be formally proved correct using **FailKAT** and some additional equations (see bottom of Fig. 1) that encode the relevant properties of the domain of computation. Reasoning in KA and KAT under the presence of additional equational hypotheses is studied in [17,24]. Our results here are general enough to apply to this setting.

Our contribution. We study within the framework of Kleene algebra with tests [13] an explicit *fail* operation, which is meant to model the abnormal termination

of imperative sequential programs, and a *try-catch* construct for handling exceptions. Our main results are the following:

- We show that every Kleene algebra with tests K can be extended conservatively with a family of $\mathsf{fail}(e)$ constants and $\mathsf{try} \ldots \mathsf{catch}(e) \ldots$ operations, where e ranges over the set E of possible exceptions. The resulting structure is the least restrictive extension of K that satisfies the axioms of **FailKAT**.
- From this conservativity theorem we obtain as a corollary a *completeness* theorem for the class of algebras of functions $X \to P(X \oplus (X \times E))$.
- The setting of **FailTKAT** requires an extended construction that works on a subclass of Kleene algebras with tests that possess a *top element* $\top$. We show how to extend conservatively such algebras with *fail* and *try-catch* operations, so that the extension is a **FailTKAT**.
- We then derive as a corollary the completeness of the calculus **FailTKAT** for the equational theory induced by the coarser semantics in terms of functions $X \to P(X \oplus E)$.

2 Relational Models of Failure

A standard denotational way of describing the meaning of (failure-free) sequential imperative programs is using binary relations or, equivalently, functions of type $X \to PX$, where X is the state space and P is the powerset functor [32]. This denotational semantics agrees with the intended operational semantics, and its advantage is that it allows us to define the meaning compositionally (by induction on the structure of the program) using certain algebraic operations on the carrier $X \to PX$. In the program semantics literature, this is usually referred to as the standard *relational semantics* of imperative programs.

In this section, we follow an analogous approach for the denotational semantics of programs that can terminate abnormally. As we have already discussed in the introduction, there are two different ways to define the meaning of such programs, which lead us to consider functions of type $X \to P(X \oplus (X \times E))$ and $X \to P(X \oplus E)$ respectively. We will endow these sets of functions with algebraic operations that are sufficient for interpreting the program structuring operations. Our treatment is infinitary, since we will be considering an arbitrary choice $\sum$ operation. This is more convenient for semantic investigations than a finitary treatment, because the language is more economical (the operations 0, $+$ and * can all be expressed in terms of $\sum$).

We write $\iota_1^{X,Y} : X \to X \oplus Y$ for the left injection map, and $\iota_2^{X,Y} : Y \to X \oplus Y$ for the right injection map. The superscript X, Y is omitted when the types can be inferred from the context. A function k of type $X \to P(X \oplus (X \times E))$ is said to be *fail-free* if $k(x)$ is contained in $\{\iota_1(x) \mid x \in X\}$ for all $x \in X$. For a set X, we define now the algebra $\mathcal{A}_X$ of all functions from X to $P(X \oplus (X \times E))$:

$$\mathcal{A}_X \triangleq (A, K, \cdot, \textstyle\sum, 1, (\mathit{fail}_e)_{e\in E}, (\mathit{try\text{-}catch}_e)_{e\in E}) \qquad 1(x) \triangleq \{\iota_1(x)\}$$
$$A \triangleq \text{functions } X \to P(X \oplus (X \times E)) \qquad \mathit{fail}_e(x) \triangleq \{\iota_2(x, e)\}$$
$$K \triangleq \{k \in A \mid k \text{ is fail-free}\} \qquad (\textstyle\sum_i f_i)(x) \triangleq \bigcup_i f_i(x)$$
$$(f \cdot g)(x) \triangleq \textstyle\bigcup\{g(y) \mid \iota_1(y) \in f(x)\} \cup \{\iota_2(y, e) \mid \iota_2(y, e) \in f(x)\}$$
$$(\mathit{try}\ f\ \mathit{catch}_e\ g)(x) \triangleq \{\iota_1(y) \mid \iota_1(y) \in f(x)\} \cup \textstyle\bigcup\{g(y) \mid \iota_2(y, e) \in f(x)\} \cup \{\iota_2(y, d) \mid \iota_2(y, d) \in f(x) \text{ and } d \neq e\}$$

Definition 1 (F-Quantales). Fix a set E of exceptions. An *F-quantale* with exceptions E is a two-sorted algebraic structure

$$(A, K, \cdot, \textstyle\sum, 1, (\mathit{fail}_e)_{e\in E}, (\mathit{try\text{-}catch}_e)_{e\in E})$$

with carriers $K \subseteq A$, where K is the sort of *fail-free elements*, and A is the sort of all elements. We require that K is closed under $\cdot$ and $\sum$, and that the following hold (the variables $u, v, \ldots$ range over A and $x, y, \ldots$ range over K):

$$(u \cdot v) \cdot w = u \cdot (v \cdot w) \qquad 1 \cdot u = 1 \qquad u \cdot 1 = 1 \tag{1}$$
$$\mathit{fail}_e \cdot u = \mathit{fail}_e \tag{2}$$
$$\textstyle\sum\{u\} = u \tag{3}$$
$$\textstyle\sum_i(\sum_j u_{ij}) = \sum_{i,j} u_{ij} \text{ (arbitrary index sets)} \tag{4}$$
$$(\textstyle\sum_i u_i) \cdot v = \sum_i u_i \cdot v \text{ (arbitrary index set)} \tag{5}$$
$$u \cdot (\textstyle\sum_i v_i) = \sum_i u \cdot v_i \text{ (}\boldsymbol{nonempty}\text{ index set)} \tag{6}$$
$$x \cdot (\textstyle\sum_i y_i) = \sum_i x \cdot y_i \text{ (arbitrary index set)} \tag{7}$$
$$\mathit{try}\ 1\ \mathit{catch}_e\ u = 1 \tag{8}$$
$$\mathit{try}\ \mathit{fail}_d\ \mathit{catch}_e\ u = \mathit{fail}_d, \text{ when } d \neq e \tag{9}$$
$$\mathit{try}\ \mathit{fail}_e\ \mathit{catch}_e\ u = u \tag{10}$$
$$\mathit{try}\ (\textstyle\sum_i u_i)\ \mathit{catch}_e\ w = \sum_i \mathit{try}\ u_i\ \mathit{catch}_e\ w \text{ (arbitrary index set)} \tag{11}$$
$$\mathit{try}\ (x \cdot v)\ \mathit{catch}_e\ w = x \cdot (\mathit{try}\ v\ \mathit{catch}_e\ w) \tag{12}$$

Lemma 2. The algebra of functions $X \to P(X \oplus (X \times E))$ is an F-quantale.

3 The Basic Algebraic Theory of Failure

In this section we investigate the basic algebraic theory of abnormal termination. One of our main goals here is to give a sound and complete axiomatization of the equational theory (in the language of KAT with fail and try-catch) of the class of relational F-quantales defined in Sect. 2. The axioms that we propose, which we call FailKAT, define a class of structures with many more interesting models other than the relational F-quantales (e.g., language models). We develop the algebraic theory of these structures. Our development consists of several steps:

– We introduce the abstract class of FailKATs. Every F-quantale (with tests), and hence every algebra of functions $X \to P(X \oplus (X \times E))$, is a model.
– We present a general model-theoretic construction that builds a FailKAT $\mathsf{F}K$ from an arbitrary KAT K. The elements of $\mathsf{F}K$ are pairs $\langle x, \psi \rangle$, where x is an element of K and $\psi : E \to K$ is an E-indexed tuple of elements of K. The component x is to be thought as the "fail-free" part, and $\psi(e)$ is the component that leads to failure with error code e.
– We show that the FailKAT $\mathsf{F}K$ can be defined equivalently in a syntactic way: expand the signature with a family of fresh constants fail_e and a family of $\mathsf{try\text{-}catch}_e$ operations, and quotient by the axioms of FailKAT.
– The aforementioned construction has several consequences, among which is the completeness of FailKAT for the theory of relational F-quantales.

Several more useful completeness results can be obtained using the results of [17,24], where free language models of KA with extra equations are identified.

$$
\begin{array}{ccc}
(x+y)+z = x+(y+z) & (x \cdot y) \cdot z = x \cdot (y \cdot z) & (x+y) \cdot z = x \cdot z + y \cdot z \\
x+y = y+x & 1 \cdot x = x & x \cdot (y+z) = x \cdot y + x \cdot z \\
x+x = x & x \cdot 1 = x & 0 \cdot x = 0 \\
x+0 = x & & x \cdot 0 = 0 \\
1 + x \cdot x^* \leq x^* \quad 1 + x^* \cdot x \leq x^* & x \cdot y \leq y \Rightarrow x^* \cdot y \leq y & y \cdot x \leq y \Rightarrow y \cdot x^* \leq y
\end{array}
$$

Fig. 2. KA: axioms for Kleene algebras [12].

A *Kleene algebra* (KA) is an algebraic structure $(K, +, \cdot, ^*, 0, 1)$ satisfying the axioms of Fig. 2, which are implicitly universally quantified. The relation $\leq$ refers to the natural partial order on K, defined as: $x \leq y \iff x + y = y$. The three top blocks of axioms (which do not involve the star operation) say that the reduct $(K, +, \cdot, 0, 1)$ is an *idempotent semiring*. We often omit the $\cdot$ operation and write xy instead of $x \cdot y$. A *Kleene algebra with tests* (KAT) is an algebraic structure $(K, B, +, \cdot, ^*, 0, 1, \neg)$ with $B \subseteq K$, satisfying the following properties: (i) the reduct $(K, +, \cdot, ^*, 0, 1)$ is a KA, (ii) B contains 0, 1 and is closed under $+$ and $\cdot$, and (iii) the reduct $(B, +, \cdot, 0, 1, \neg)$ is a Boolean algebra.

Definition 3 (FailKAT). Fix a set E of exceptions. A *KAT with failures* E, or simply *FailKAT*, is a three-sorted algebra

$$(A, K, B, +, \cdot, ^*, 0, 1, \neg, (\mathit{fail}_e)_{e \in E}, (\mathit{try\text{-}catch}_e)_{e \in E})$$

with $B \subseteq K \subseteq A$, where K is closed under $+$, $\cdot$ and *, and $(K, B, +, \cdot, ^*, 0, 1, \neg)$ is a KAT. Moreover, $(A, +, \cdot, ^*, 0, 1, \mathit{fail}_e, \mathit{try\text{-}catch}_e)$ satisfies the axioms of KA except for $u \cdot 0 = 0$, and it also satisfies the *fail* axiom (2) and the *try-catch* axioms (8)–(12) of Definition 1. We say that K is the carrier of *fail-free* elements, and A is the carrier of all elements (which may allow failure).

For a KAT K and a set E of exceptions, we denote by K^E the set of E-indexed tuples (equivalently, the set of functions $E \to K$). The operation $+$ is defined on K^E componentwise: $(\phi + \psi)(e) = \phi(e) + \psi(e)$. For $x \in K$ and $\phi \in K^E$, we define $x \cdot \phi$ as follows: $(x \cdot \phi)(e) = x \cdot \phi(e)$. For all elements $x, y \in K$ and every tuple $\phi \in K^E$, the distributivity property $(x + y) \cdot \phi = (x \cdot \phi) + (y \cdot \phi)$ holds. The zero tuple $\bar{0}$ is defined as $\bar{0}(e) = 0$ for all e. For a tuple $\phi \in K^E$, an exception e, and an element $x \in K$, we write $\phi[x/e]$ for the tuple that agrees with ϕ on $E \setminus \{e\}$ and whose e-th component is equal to x. We say that a tuple is of *finite support* if it has finitely many non-zero components. We write $K^{\langle E \rangle}$ for the set of all tuples of K^E that have finite support.

Definition 4 (The Construction F). Let $(K, B, +, \cdot, ^*, 0, 1, \neg)$ be a KAT, and E a set of exceptions. We define the three-sorted algebra

$$\mathsf{F}K \triangleq (K \times K^{\langle E \rangle}, K \times \{\bar{0}\}, B \times \{\bar{0}\}, +, \cdot, ^*, 0^{\mathsf{F}}, 1^{\mathsf{F}}, \neg, \mathit{fail}_e^{\mathsf{F}}, \mathit{try\text{-}catch}_e^{\mathsf{F}})$$

with carriers $K \times K^{\langle E \rangle}$, $K \times \{\bar{0}\}$ and $B \times \{\bar{0}\}$, as follows:

$$\begin{aligned}
0^{\mathsf{F}} &\triangleq \langle 0, \bar{0} \rangle & \langle x, \phi \rangle + \langle y, \psi \rangle &\triangleq \langle x + y, \phi + \psi \rangle \\
1^{\mathsf{F}} &\triangleq \langle 1, \bar{0} \rangle & \langle x, \phi \rangle \cdot \langle y, \psi \rangle &\triangleq \langle x \cdot y, \phi + x \cdot \psi \rangle \\
\mathit{fail}_e^{\mathsf{F}} &\triangleq \langle 0, \bar{0}[1/e] \rangle & \langle x, \phi \rangle^* &\triangleq \langle x^*, x^* \cdot \phi \rangle \\
\neg \langle p, \bar{0} \rangle &\triangleq \langle \neg p, \bar{0} \rangle & \mathit{try}\ \langle x, \phi \rangle\ \mathit{catch}_e\ \langle y, \psi \rangle &\triangleq \langle x + \phi(e) \cdot y, \phi[0/e] + \phi(e) \cdot \psi \rangle
\end{aligned}$$

Informally, the idea is that an element $\langle x, \phi \rangle$ of $\mathsf{F}K$ consists of fail-free component x, and the component $\phi(e)$ which leads to failure with error code e. From the definition of $+$ we get that $\langle x, \phi \rangle \leq \langle y, \psi \rangle$ iff $x \leq y$ and $\phi(e) \leq \psi(e)$ for all e.

Definition 4 is inspired from the operational intuition of how programs with exceptions compute. Assuming that there is only one exception, we think of a pair $\langle x, \phi \rangle$ in $\mathsf{F}K$ as the program $x + \phi \cdot \mathsf{fail}$. The operation $\cdot$ in $\langle x, \phi \rangle \cdot \langle y, \psi \rangle$ models sequential composition. The fail-free component of $\langle x, \phi \rangle \cdot \langle y, \psi \rangle$ corresponds to the possibility of executing x and y in sequence, and failure can result by either executing ϕ or by executing x and ψ in sequence. The definitions of the rest of the operations can be understood similarly.

Lemma 5. Let K be a KAT and E be a set of exceptions. The algebra $\mathsf{F}K$ is a FailKAT, and the map $x \mapsto \langle x, \bar{0} \rangle$ is a KAT embedding of K into $\mathsf{F}K$.

Definition 6 (Adjoin Elements for Failure). Let $(K, B, +, \cdot, ^*, 0, 1, \neg)$ be a KAT, E be a set of exceptions, and fail_e for all $e \in E$ be fresh distinct symbols that denote failure or abnormal termination. We also consider for every exception e a fresh binary operation symbol $\mathsf{try\text{-}catch}_e$. For every element $x \in K$ we introduce a constant symbol c_x. We define the sets $\mathsf{Trm}_B(K) \subseteq \mathsf{Trm}(K) \subseteq \mathsf{Trm}_F(K)$ of algebraic terms with the following generation rules:

$$\frac{p \in B}{c_p \in \mathsf{Trm}_B(K)} \qquad \frac{s,t \in \mathsf{Trm}_B(K)}{s+t,\ s\cdot t,\ \neg s \in \mathsf{Trm}_B(K)} \qquad \frac{t \in \mathsf{Trm}_B(K)}{t \in \mathsf{Trm}(K)}$$

$$\frac{x \in K}{c_x \in \mathsf{Trm}(K)} \qquad \frac{s,t \in \mathsf{Trm}(K)}{s+t,\ s\cdot t,\ t^* \in \mathsf{Trm}(K)} \qquad \frac{t \in \mathsf{Trm}(K)}{t \in \mathsf{Trm}_F(K)}$$

$$\mathsf{fail}_e \in \mathsf{Trm}_F(K) \qquad \frac{s,t \in \mathsf{Trm}_F(K)}{s+t,\ s\cdot t,\ t^*, \mathsf{try}\, s\, \mathsf{catch}_e\, t \in \mathsf{Trm}_F(K)}$$

The function $c_x \mapsto x$, where $x \in K$, extends uniquely to a homomorphism $k : \mathsf{Trm}(K) \to K$. The *diagram* of K, defined as $\Delta_K \triangleq \{s \equiv t \mid k(s) = k(t)\}$, is the set of equations $s \equiv t$ that are true under k. In other words, Δ_K is the *kernel* of the homomorphism k. Finally, we define the set of equations

$$E_K \triangleq \text{FailKAT-Closure}(\Delta_K)$$

to be the least set that contains Δ_K and is closed under the axioms and rules of FailKAT and Horn-equational logic. By the axioms and rules of equational logic, the equations of E_K define a FailKAT-congruence on $\mathsf{Trm}_F(K)$. For a term t in $\mathsf{Trm}_F(K)$, we write $[t]_E$ to denote its congruence class. Define the three-sorted algebra $\mathbb{F}K$ with carriers $\hat{B} \subseteq \hat{K} \subseteq A$ as:

$$A \triangleq \{[t]_E \mid t \in \mathsf{Trm}_F(K)\} \quad \hat{K} \triangleq \{[t]_E \mid t \in \mathsf{Trm}(K)\} \quad \hat{B} \triangleq \{[t]_E \mid t \in \mathsf{Trm}_B(K)\}$$

Since the equations E_K define a FailKAT-congruence, we can define the FailKAT operations of $\mathbb{F}K$ on the equivalence classes of terms:

$$0^{\mathbb{F}} \triangleq [c_0]_E \qquad \mathit{fail}_e^{\mathbb{F}} \triangleq [\mathsf{fail}_e]_E \qquad [s]_E + [t]_E \triangleq [s+t]_E \qquad ([t]_E)^* \triangleq [t^*]_E$$
$$1^{\mathbb{F}} \triangleq [c_1]_E \qquad\qquad [s]_E \cdot [t]_E \triangleq [s\cdot t]_E$$

We have thus defined the algebra $\mathbb{F}K$, which has the signature of FailKATs.

Lemma 7 (Normal Form). For every term t in $\mathsf{Trm}_F(K)$ there are fail-free terms t_P and t_e in $\mathsf{Trm}(K)$ (for every exception e that appears in t) such that the equation $t \equiv t_P + \sum_e t_e \cdot \mathsf{fail}_e$ is in E_K.

Proof. Since terms are finite and only finitely many exceptions occur in them, we fix w.l.o.g. a *finite* E. The proof is by induction on the structure of t. If t is a fail-free term, then notice that the equation $t \equiv t + \sum_{e\in E} 0 \cdot \mathsf{fail}_e$ is in E_K. For the case $t = \mathsf{fail}_d$, we observe that $\mathsf{fail}_d \equiv 0 + 1 \cdot \mathsf{fail}_d + \sum_{e\neq d} 0 \cdot \mathsf{fail}_e$ is in E_K. For the induction step, we have the following equations in E_K:

$$\begin{aligned} s + t &\equiv (s_P + \textstyle\sum_e s_e \cdot \mathsf{fail}_e) + (t_P + \sum_e t_e \cdot \mathsf{fail}_e) \\ &\equiv (s_P + t_P) + \textstyle\sum_{e\in E}(s_e + t_e) \cdot \mathsf{fail}_e \\ s \cdot t &\equiv (s_P + \textstyle\sum_e s_e \cdot \mathsf{fail}_e) \cdot (t_P + \sum_e t_e \cdot \mathsf{fail}_e) \\ &\equiv s_P \cdot t_P + \textstyle\sum_e s_P \cdot t_e \cdot \mathsf{fail}_e + \sum_e s_e \cdot \mathsf{fail}_e \cdot (\ldots) \\ &\equiv s_P \cdot t_P + \textstyle\sum_e (s_e + s_P \cdot t_e) \cdot \mathsf{fail}_e \end{aligned}$$

$$
\begin{aligned}
t^* &\equiv (t_P + \textstyle\sum_e t_e \cdot \mathsf{fail}_e)^* \\
&\equiv t_P^* \cdot (\textstyle\sum_e t_e \cdot \mathsf{fail}_e \cdot t_P^*)^* \\
&\equiv t_P^* \cdot (\textstyle\sum_e t_e \cdot \mathsf{fail}_e)^* \\
&\equiv t_P^* \cdot (1 + (\textstyle\sum_e t_e \cdot \mathsf{fail}_e) \cdot (\textstyle\sum_e t_e \cdot \mathsf{fail}_e)^*) \\
&\equiv t_P^* \cdot (1 + \textstyle\sum_e t_e \cdot \mathsf{fail}_e) \\
&\equiv t_P^* + \textstyle\sum_e t_P^* \cdot t_e \cdot \mathsf{fail}_e \\
\mathsf{try}\, s\, \mathsf{catch}_d\, t &\equiv \mathsf{try}\, (s_P + \textstyle\sum_e s_e \cdot \mathsf{fail}_e)\, \mathsf{catch}_d\, (t_P + \textstyle\sum_e t_e \cdot \mathsf{fail}_e) \\
&\equiv \mathsf{try}\, s_P\, \mathsf{catch}_d\, (t_P + \textstyle\sum_e t_e \cdot \mathsf{fail}_e) \\
&\quad + \textstyle\sum_e \mathsf{try}\, (s_e \cdot \mathsf{fail}_e)\, \mathsf{catch}_d\, (t_P + \textstyle\sum_e t_e \cdot \mathsf{fail}_e) \\
&\equiv s_P \cdot (\mathsf{try}\, 1\, \mathsf{catch}_d\, (t_P + \textstyle\sum_e t_e \cdot \mathsf{fail}_e)) \\
&\quad + \textstyle\sum_e s_e \cdot (\mathsf{try}\, \mathsf{fail}_e\, \mathsf{catch}_d\, (t_P + \textstyle\sum_e t_e \cdot \mathsf{fail}_e)) \\
&\equiv s_P + s_d \cdot (t_P + \textstyle\sum_e t_e \cdot \mathsf{fail}_e) + \textstyle\sum_{e \neq d} s_e \cdot \mathsf{fail}_e \\
&\equiv (s_P + s_d \cdot t_P) + \textstyle\sum_e s_d \cdot t_e \cdot \mathsf{fail}_e + \textstyle\sum_{e \neq d} s_e \cdot \mathsf{fail}_e \\
&\equiv (s_P + s_d \cdot t_P) + s_d \cdot t_d \cdot \mathsf{fail}_d + \textstyle\sum_{e \neq d} (s_e + s_d \cdot t_e) \cdot \mathsf{fail}_e
\end{aligned}
$$

We have used the property $(x + y)^* = x^*(yx^*)^*$, which is a theorem of KA. □

Theorem 8 (F and 𝔽). The FailKATs $\mathsf{F}K$ and $\mathbb{F}K$ are isomorphic.

Proof. We define the map $h : \mathsf{Trm}_F(K) \to \mathsf{F}K$ to be the unique homomorphism satisfying $h(c_x) = \langle x, \bar{0} \rangle$ and $h(\mathsf{fail}_e) = \langle 0, \bar{0}[1/e] \rangle$. Notice that h sends fail-free terms to fail-free elements of $\mathsf{F}K$. By Lemma 5, $\mathsf{F}K$ is a FailKAT, and therefore $h(s) = h(t)$ for every equation $s \equiv t$ in E_K. So, we can define the map $\hat{h} : \mathsf{Trm}_F(K)/E_K \to \mathsf{F}K$ by $[t]_E \mapsto h(t)$. In fact, $\hat{h}$ is a FailKAT homomorphism from $\mathbb{F}K$ to $\mathsf{F}K$. Moreover, $\hat{h}$ is surjective: for every $\langle x, \phi \rangle \in \mathsf{F}K$, where $D \subseteq E$ is the finite support of ϕ, we have that

$$
\begin{aligned}
\hat{h}([c_x + \textstyle\sum_{e \in D} c_{\phi(e)} \cdot \mathsf{fail}_e]_E) &= h(c_x + \textstyle\sum_{e \in D} c_{\phi(e)} \cdot \mathsf{fail}_e) \\
&= h(c_x) + \textstyle\sum_{e \in D} h(c_{\phi(e)}) \cdot h(\mathsf{fail}_e) \\
&= \langle x, \bar{0} \rangle + \textstyle\sum_{e \in D} \langle \phi(e), \bar{0} \rangle \cdot \langle 0, \bar{0}[1/e] \rangle \\
&= \langle x, \bar{0} \rangle + \textstyle\sum_{e \in D} \langle 0, \phi(e) \cdot \bar{0}[1/e] \rangle \\
&= \langle x, \phi \rangle.
\end{aligned}
$$

Finally, we claim that $\hat{h}$ is injective. Suppose that $\hat{h}([s]_E) = \hat{h}([t]_E)$. Lemma 7 says that there are fail-free terms s_P, s_e, t_P, t_e (for $e \in D \subseteq E$) in $\mathsf{Trm}(K)$ s.t.

$$
s \equiv s_P + \textstyle\sum_e s_e \cdot \mathsf{fail}_e \qquad\qquad t \equiv t_P + \textstyle\sum_e t_e \cdot \mathsf{fail}_e
$$

are equations of E_K. From our hypothesis above we obtain that

$$
\begin{aligned}
\hat{h}([s]_E) = \hat{h}([t]_E) &\implies h(s) = h(t) \\
&\implies h(s_P + \textstyle\sum_e s_e \cdot \mathsf{fail}_e) = h(t_P + \textstyle\sum_e t_e \cdot \mathsf{fail}_e) \\
&\implies \langle k(s_P), (k(s_e))_{e \in D} \rangle = \langle k(t_P), (k(t_e))_{e \in D} \rangle \\
&\implies k(s_P) = k(t_P) \text{ and } k(s_e) = k(t_e) \text{ for all } e \in D.
\end{aligned}
$$

(Recall the function $k : \mathsf{Trm}(K) \to K$ from Definition 6.) It follows that the equations $s_P \equiv t_P$ and $s_e \equiv t_e$ are in the diagram Δ_K. So, $s \equiv t$ is in E_K. We thus obtain that $[s]_E = [t]_E$. So, $\hat{h}$ is a FailKAT isomorphism. □

The above theorem says that the extension of a KAT K with fail_e elements and $\mathsf{try\text{-}catch}_e$ operations is *conservative*, since the mapping $x \mapsto \langle x, \bar{0}\rangle$ embeds K into $\mathbb{F}K \cong \mathsf{F}K$. This is a very useful property, because it means that a language of while-programs (whose semantics is defined by interpretation in a KAT) can be extended naturally to accomodate the extra programming feature of failure.

Corollary 9 (Completeness for Relational Models). FailKAT is complete for the equational theory of the class of algebras $X \to P(X \oplus (X \times E))$.

Proof. It is known from [12,13] that KA and KAT are complete for the class of relational models $X \to PX$. In fact, for a fixed finite set Σ of atomic actions and a fixed finite set B_0 of atomic tests, there is a single full relational model $\mathrm{Rel}(\Sigma, B_0)$ that characterizes the theory. We fix a finite set E of exceptions. Using the observation $P(X \oplus (X \times E)) \cong (PX) \times (PX)^E$ we can see that the algebras $X \to P(X \oplus (X \times E))$ and $\mathsf{F}(X \to PX)$ (recall Definition 4) are isomorphic. Since KAT is complete for the theory of $\mathrm{Rel}(\Sigma, B_0)$, Theorem 8 implies that FailKAT is complete for the theory of $\mathsf{FRel}(\Sigma, B_0)$. This means that FailKAT is complete for the class of all relational models. □

Let Σ be a finite set of atomic actions, and B_0 be a finite set of atomic tests. Kozen and Smith have shown in [20] that KAT is complete for $\mathsf{Reg}(\Sigma, B_0)$, the algebra of regular sets of guarded strings over Σ and B_0. Theorem 8 implies that FailKAT is complete for the algebra $\mathsf{FReg}(\Sigma, B_0)$, which is therefore the free FailKAT with action generators Σ and test generators B_0.

First, we observe that the decidability of FailKAT is an easy consequence of Lemma 7. To decide the equivalence of two terms s and t, we rewrite them according to the proof of Lemma 7 into equivalent normal forms $s \equiv s_P + \sum_e s_e \cdot \mathsf{fail}_e$ and $t \equiv t_P + \sum_e t_e \cdot \mathsf{fail}_e$, where e ranges over the exceptions that appear in s and t and all the terms s_P, s_e, t_P, t_e are fail-free. It follows that $s \equiv t$ iff $s_P \equiv t_P$ and $s_e \equiv t_e$ for all e, hence equivalence can be decided using a decision procedure for KAT. As discussed in the previous paragraph, the language model $\mathsf{FReg}(\Sigma, B_0)$ characterizes the equational theory of FailKAT, which suggests that an appropriate variant of Kozen's guarded automata [14,16] can be used to decide the equational theory in polynomial space.

Example 10. Exceptions and their handlers can be used to encode widely used constructs of non-local control flow, such as break and continue. The program

$$h = \mathsf{while}\,(\mathit{true})\ \mathsf{do}\,\{a;\ (\mathsf{if}\ \bar{p}\ \mathsf{then}\ \mathsf{break});\ (\mathsf{if}\ \bar{q}\ \mathsf{then}\ \mathsf{continue});\ b\}$$

can be shown to be equivalent to $h' = a;\, \mathsf{while}\, p\ \mathsf{do}\,\{(\mathsf{if}\ q\ \mathsf{then}\ b);\, a\}$ using FailKAT. We abbreviate $\neg p$ by $\bar{p}$, and $\neg q$ by $\bar{q}$. The program h is encoded as follows:

```
try {
    while (true) do {
        try {a; (if p̄ then fail_e); (if q̄ then fail_d); b} catch(d) { }
    }
} catch(e) { }
```

Let g be the body of the while loop, and f be subprogram of the inner try-catch statement. In the language of FailKAT, we have that $f = a(\bar{p}fail_e + p)(\bar{q}fail_d + q)b$, $g = try\ f\ catch_d\ 1$, and $h = try\ (g^*0)\ catch_e\ 1$. Using the FailKAT axioms we get:

$$\begin{aligned}
f &= a\bar{p}fail_e + ap(\bar{q}fail_d + q)b = a\bar{p}fail_e + ap\bar{q}fail_d + apqb \\
g &= a\bar{p}fail_e + ap\bar{q} + apqb \\
g^* &= ((ap\bar{q} + apqb) + a\bar{p}fail_e)^* = (ap\bar{q} + apqb)^*(a\bar{p}fail_e(ap\bar{q} + apqb)^*)^* \\
&= (ap\bar{q} + apqb)^*(a\bar{p}fail_e)^* = (ap\bar{q} + apqb)^*(1 + a\bar{p}fail_e) \\
g^*0 &= (ap\bar{q} + apqb)^*(1 + a\bar{p}fail_e)0 = (ap\bar{q} + apqb)^*a\bar{p}fail_e \\
h &= (ap\bar{q} + apqb)^*a\bar{p} = (a(p\bar{q} + pqb))^*a\bar{p} \\
h' &= a(p(qb + \bar{q})a)^*\bar{p} = a((pqb + p\bar{q})a)^*\bar{p} = (a(pqb + p\bar{q}))^*a\bar{p}
\end{aligned}$$

which means that $h = h'$. We have used above the theorems $(x+y)^* = x^*(yx^*)^*$ and $x(yx)^* = (xy)^*x$ of KA.

Example 11. We will establish using FailKAT the equivalence of the programs of Fig. 1. The program on the right-hand side is an optimized version because it eliminates the check of the condition $i \geq 0$. To streamline the presentation we abbreviate $(i := 0)$ by a, $(X[i] := 0)$ by b, and $(i := i + 1)$ by c. We also use the abbreviations p, q and r for the tests $(i < n)$, $(0 \leq i)$ and $(i < X.\texttt{length})$ respectively, and write $\bar{p}$, $\bar{q}$, $\bar{r}$ instead of $\neg p$, $\neg q$, $\neg r$ respectively. The extra hypotheses of Fig. 1 can be then written as $a = aq$, $qb = bq$, and $qc = qcq$. Using these abbreviations we encode the right-hand side program of Fig. 1 as $R = a(p(\bar{r}fail + r)bc)^*\bar{p} = a(p\bar{r}fail + prbc)^*\bar{p} = ag^*\bar{p}$, where $g = prbc + p\bar{r}fail$, and the left-hand side program as

$$L = a(p(qr + \neg(qr)fail)bc)^*\bar{p} = a(p(qr + (\bar{q} + q\bar{r})fail)bc)^*\bar{p} = ah^*\bar{p},$$

where $h = pqrbc + p\bar{q}fail + pq\bar{r}fail$. With $qb = bq$ and $qc = qcq$ we show $qh = qhq$ and hence $qh^* = q(qhq)^* = q(qh)^* = q(pqrbc + pq\bar{r}fail)^* = q(q(prbc + p\bar{r}fail))^*$. Since $qg = qgq$ we obtain similarly that $qh^* = q(qg)^* = qg^*$. Finally, using the hypothesis $a = aq$ we obtain that $L = ah^*\bar{p} = aqh^*\bar{p} = aqg^*\bar{p} = ag^*\bar{p} = R$.

4 A Stronger Theory of Failure

In this section we investigate a stronger (i.e., with more theorems) algebraic theory of abnormal termination. We will write $\approx$ to refer to this notion of equivalence to differentiate it from the weaker equivalence that we studied in Sect. 3.

This stronger theory, called **FailTKAT**, results from FailKAT by adding the axioms

$$u = v \implies u \approx v \qquad x \cdot \mathit{fail}_e \lesssim \mathit{fail}_e \qquad \approx \text{ is KA-congruence}$$

where u, v are arbitrary elements and x is a fail-free element. As in the previous section, our ultimate goal is to give a sound and complete axiomatization of the relation $\approx$ on the algebra $X \to P(X \oplus (X \times E))$. First, we define a projection operation π that "forgets" the output state in the case or error:

$$\frac{f : X \to P(X \oplus (X \times E))}{\pi(f) : X \to P(X \oplus E)} \qquad \begin{aligned} \pi(f)(x) \triangleq\ & \{\iota_1(y) \mid \iota_1(y) \in f(x)\} \cup \\ & \{\iota_2(e) \mid \iota_2(y, e) \in f(x)\} \end{aligned}$$

The $\approx$ equivalence can then be defined as follows: $f \approx g$ iff $\pi(f) = \pi(g)$.

The operation of forgetting the output state is defined for algebras of input/output relations in the obvious way, but it is not apparent if a more general construction can be formulated for a subclass of Kleene algebras. As it turns out, there exists a very natural subclass of KATs, which we call KATs with a top element, for which this is possible.

Definition 12 (KAT With Top Element). A *KAT with a top element* or a *TopKAT* is a structure $(K, B, +, \cdot, ^*, 0, 1, \neg, \top)$ so that $(K, B, +, \cdot, ^*, 0, 1, \neg)$ is a KAT and the *top element* $\top$ satisfies the inequality $x \leq \top$ for all $x \in K$.

Intuitively, the top element $\top$ is needed to forget the state of the memory. More precisely, right multiplication $(- \cdot \top)$ by the top element models the projection function that eliminates the state. Without the top element we cannot define the coarser equivalence relation $\approx$ using the operations of KA.

Definition 13 (Generalized $\approx$ Equivalence). Let K be a TopKAT and E be a set of exceptions. We define for the algebra $\mathsf{F}K$ of Definition 4 the equivalence relation $\approx$ as follows: $\langle x, \phi\rangle \approx \langle y, \psi\rangle$ iff

$$x = y \text{ and } \phi(e) \cdot \top = \psi(e) \cdot \top \text{ for every } e \in E.$$

The *projection map* $\pi : \langle x, \phi\rangle \mapsto \langle x, \phi'\rangle$ is defined as $\phi'(e) = \phi(e) \cdot \top$ for all e. So, $\approx$ can be equivalently defined as: $\langle x, \phi\rangle \approx \langle y, \psi\rangle$ iff $\pi(\langle x, \phi\rangle) = \pi(\langle y, \psi\rangle)$.

Lemma 14 (Projection). Let K be a TopKAT and E be a set of exceptions. For the algebra $\mathsf{F}K$ and the projection map π of Definition 13 the following hold:

$$\begin{aligned} &\pi(\mathit{fail}_e^{\mathsf{F}}) = \langle 0, \bar{0}[\top/e]\rangle && \pi(1^{\mathsf{F}}) = 1^{\mathsf{F}} && \pi(0^{\mathsf{F}}) = 0^{\mathsf{F}} \\ &\pi(u + v) = \pi(v) + \pi(v) && \pi(u \cdot v) = \pi(u) \cdot \pi(v) && \pi(u^*) = \pi(u)^* \end{aligned}$$

Moreover, $\pi(x \cdot \mathit{fail}_e^{\mathsf{F}} + \mathit{fail}_e^{\mathsf{F}}) = \pi(\mathit{fail}_e^{\mathsf{F}})$ or, equivalently, $x \cdot \mathit{fail}_e^{\mathsf{F}} \lesssim \mathit{fail}_e^{\mathsf{F}}$. *Note*: the variables u, v range over arbitrary elements and x ranges over fail-free elements.

Proof. The commutation properties are easy to verify. For the second part, we have: $\pi(\langle 0, \bar{0}[x/e]\rangle + \langle 0, \bar{0}[1/e]\rangle) = \pi(\langle 0, \bar{0}[x+1/e]\rangle) = \langle 0, \bar{0}[(x+1)\top/e]\rangle$ and also $\pi(\langle 0, \bar{0}[1/e]\rangle) = \langle 0, \bar{0}[\top/e]\rangle$. It suffices to show that $(x+1)\top = \top$ in the TopKAT K, which is true because $(x+1)\top = x\top + \top = \top$. □

The equations of Lemma 14 that show how π commutes with the KA operations of $\mathsf{F}K$ imply additionally that $\approx$ is a KA-congruence. For example, $u \approx v$ implies $\pi(u) = \pi(v)$, which gives us $\pi(u^*) = \pi(u)^* = \pi(v)^* = \pi(v^*)$ and therefore $u^* \approx v^*$. So, $\mathsf{F}K$ with $\approx$ is a model of the FailTKAT axioms.

We extend now Definition 6 to expand the algebra $\mathbb{F}K$, where K is a TopKAT, with a relation $\approx$. Similarly to the construction of the previous section, $\approx$ is given as follows for terms s and t: $[s]_E \approx [t]_E$ iff $s \approx t$ is provable using the system FailTKAT and the diagram of K.

Theorem 15. Let K be a TopKAT and E be a set of exceptions. The FailTKATs $(\mathsf{F}K, \approx)$ and $(\mathbb{F}K, \approx)$ are isomorphic.

Proof. The proof extends the one for Theorem 8. It remains to show that for all terms s and t: $[s]_E \approx [t]_E$ iff $\hat{h}([s]_E) \approx \hat{h}([t]_E)$. The right-to-left direction is the interesting one. By Lemma 7, we bring s and t to their normal forms and:

$$\begin{aligned}
\hat{h}([s]_E) \approx \hat{h}([t]_E) &\implies \pi(h(s)) = \pi(h(t)) \\
&\implies \pi(h(s_P + \textstyle\sum_e s_e \cdot \mathsf{fail}_e)) = \pi(h(t_P + \sum_e t_e \cdot \mathsf{fail}_e)) \\
&\implies \langle k(s_P), (k(s_e)\top)_{e\in D}\rangle = \langle k(t_P), (k(t_e)\top)_{e\in D}\rangle \\
&\implies k(s_P) = k(t_P) \text{ and } k(s_e \cdot c_\top) = k(t_e \cdot c_\top) \text{ for all } e \in D.
\end{aligned}$$

It follows that $s_P \equiv t_P$ and $s_e \cdot c_\top \equiv t_e \cdot c_\top$ are in the diagram of K. Now,

$$\begin{aligned}
s &\approx s_P + \textstyle\sum_e s_e \cdot \mathsf{fail}_e \approx s_P + \sum_e s_e \cdot c_\top \cdot \mathsf{fail}_e \\
&\approx t_P + \textstyle\sum_e t_e \cdot c_\top \cdot \mathsf{fail}_e \approx t_P + \sum_e t_e \cdot \mathsf{fail}_e \approx t
\end{aligned}$$

is provable, because $x \cdot \mathsf{fail}_e \approx x \cdot c_\top \cdot \mathsf{fail}_e$ follows from Δ_K and FailTKAT.

$$1 \lesssim c_\top \Rightarrow x \cdot \mathsf{fail}_e \lesssim x \cdot c_\top \cdot \mathsf{fail}_e \qquad c_\top \cdot \mathsf{fail}_e \lesssim \mathsf{fail}_e \Rightarrow x \cdot c_\top \cdot \mathsf{fail}_e \lesssim x \cdot \mathsf{fail}_e$$

We thus obtain that $[s]_E \approx [t]_E$. So, $\hat{h}$ is a FailTKAT isomorphism. □

Similarly to Theorem 8 of the previous section, we interpret Theorem 15 as saying that an arbitrary KAT with a top element can be *extended conservatively* into a KAT with failure that satisfies the additional axiom $x \cdot \mathit{fail}_e \lesssim \mathit{fail}_e$. The mapping $x \mapsto \langle x, \bar{0}\rangle$ embeds the TopKAT K into the extension $(\mathsf{F}K, \approx)$.

Corollary 16 (Completeness for Relational Models). FailTKAT is complete for the theory of the relation $\approx$ on the class of algebras $X \to P(X \oplus (X \times E))$. *Notation*: Recall that $f \approx g$ iff the projections of f and g to functions of type $X \to P(X \oplus E)$ (by applying π) are equal.

Proof (sketch). Similar to the proof of Corollary 9. We have to show that the first-order structures $X \to P(X \oplus (X \times E))$ and $\mathsf{F}(X \to PX)$ (signature extended with the projection π and the equivalence $\approx$) are isomorphic. This relies on the observation $P(X \oplus E) \cong (PX) \times (\mathbb{1} \oplus \mathbb{1})^E \cong (PX) \times (\{\emptyset\} \oplus \{X\})^E$. □

In the previous section we discussed how our conservativity result for FailKAT gives as easy consequences the existence of a free language model, the decidability of the theory, and also suggests a way to approach the question of complexity using guarded automata. The situation is similar for FailTKAT, it suffices to observe that the free KAT $\mathsf{Reg}(\Sigma, B_0)$ with action generators Σ and test generators B_0 has a top element: the guarded language denoted by the expression Σ^*.

5 Related Work

As far as the basic theory of failure is concerned, the works [9,15] are closely related to ours. In both these papers, extensions of KAT are investigated that can be used for reasoning about nonlocal flow of control, using e.g. labels and goto statements. Syntactically, these systems amount essentially to using matrices of expressions, where the row index corresponds to an entry label and the column index corresponds to an exit label. While the fail operation can be encoded using such general constructs of nonlocal flow of control, the works [9,15] do not address the question of whether it is possible to axiomatize fail directly, i.e. without translation into a more complicated language. We have shown here that this is indeed the case, which is a new result and does not follow from any of [9,15]. The system FailKAT that we introduce here axiomatizes the properties of failure directly, and thus offers a more convenient style of reasoning for this computational phenomenon. More importantly, we ***depart completely*** from the investigations of [9,15] when we consider the stronger theory FailTKAT. None of these earlier systems can capture the properties of failure under the coarser equivalence that we study here.

Aceto and Hennessy study in [1,2] a process algebra that includes an explicit symbol δ for *deadlock*. This is somewhat similar to our fail operation, since δ satisfies the equational property $\delta; x = \delta$. Our work is, however, markedly distinct and contributes very different results. The work of Aceto and Hennessy studies a notion of bisimulation preorder, which does not have the same theory as language or trace equivalence. We axiomatize here the input-output behavior of programs, which correponds to language (and not bisimulation) equivalence.

The literature on computational effects, monads [8,28], and algebraic operations [11,30,31] is somewhat related to our work at a conceptual level. Within this body of work, exceptions and failure are modeled using the formalism of monads. In sharp contrast to what we are doing here, the work on monads typically focuses on the type structure (whereas we have here only two sorts!) and different program structuring operations (e.g., products and function abstraction) in the setting of a functional language. At a technical level, there is not

much of an intersection between our investigations and the work on monads. The language of KAT is generally more abstract than the monad-based formalisms, which maybe include constructs like products, as in $A \times B \to P(A \times B)$, or return values, as in $A \to (S \to P(S \times B))$. Such extra constructs can easily make it impossible to obtain any kind of useful completeness theorem. In particular, if the language allows loops and binary products, then we can encode abstractions of imperative programs whose state can be decomposed in variables that can be read from and written to independently. This is the case of the so-called "two-variable while program schemes", whose partial-correctness and equational theory are *not recursively enumerable* [22]. This suggests that the abstraction level of KA/KAT is necessary for obtaining meaningful unconditional completeness theorems for programs with an iteration construct.

6 Conclusion

We have considered here two algebraic theories, called FailKAT and FailTKAT, for imperative while programs with an explicit fail operation that causes abnormal termination. The system FailKAT captures the notion of program equivalence that results from a semantics that allows for the observation of the final state upon failure. The system FailTKAT captures a coarser notion of equivalence, namely when we cannot observe the final state of the computation upon failure. Both notions of equivalence are meaningful and useful, and we have seen that they admit simple and intuitive axiomatizations. From a technical perspective, the case of FailTKAT is more challenging and interesting.

A important direction for future work is the study of FailKAT and FailTKAT in the coalgebraic setting. Such an investigation would contribute to the question (posed by Kozen in [15]) of whether there is a simple coalgebraic treatment of nonlocal flow of control involving a definition of derivatives [4,5] for the nonlocal control flow constructs. We expect that the *fail* and *try-catch* constructs, which are much more structured than labels and goto statements, lend themselves to an elegant coalgebraic treatment. At a practical level, this would give rise to simple and efficient automata-theoretic decision procedures.

Another interesting question is whether the ideas of the present paper can be applied to other logical systems. Some apparent candidates are variations of KAT such as NetKAT [3,7] and Nominal KA [18,19]. Abnormal termination and nonlocal flow of control have been studied in the context of partial correctness theories based on Hoare logic (see, for example, [6,10,29,33,34]). It seems likely that the axioms of FailKAT can inspire axioms and rules for Hoare logics that treat failure and exception handling, and even obtain unconditional completeness results (see, for example, [21,23,25–27]) in the propositional setting.

References

1. Aceto, L., Hennessy, M.: Termination, deadlock and divergence. In: Main, M., Melton, A., Mislove, M., Schmidt, D. (eds.) MFPS 1989. LNCS, vol. 442, pp. 301–318. Springer, New York (1990). doi:10.1007/BFb0040264
2. Aceto, L., Hennessy, M.: Termination, deadlock, and divergence. J. ACM **39**(1), 147–187 (1992)
3. Anderson, C.J., Foster, N., Guha, A., Jeannin, J.B., Kozen, D., Schlesinger, C., Walker, D.: NetKAT: semantic foundations for networks. In: Proceedings of the 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2014), pp. 113–126 (2014)
4. Antimirov, V.: Partial derivatives of regular expressions and finite automaton constructions. Theor. Comput. Sci. **155**(2), 291–319 (1996)
5. Brzozowski, J.A.: Derivatives of regular expressions. J. ACM **11**(4), 481–494 (1964)
6. Delbianco, G.A., Nanevski, A.: Hoare-style reasoning with (algebraic) continuations. In: Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming (ICFP 2013), pp. 363–376 (2013)
7. Foster, N., Kozen, D., Mamouras, K., Reitblatt, M., Silva, A.: Probabilistic NetKAT. In: Thiemann, P. (ed.) ESOP 2016. LNCS, vol. 9632, pp. 282–309. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49498-1_12
8. Goncharov, S., Schröder, L., Mossakowski, T.: Kleene monads: handling iteration in a framework of generic effects. In: Kurz, A., Lenisa, M., Tarlecki, A. (eds.) CALCO 2009. LNCS, vol. 5728, pp. 18–33. Springer, Heidelberg (2009). doi:10.1007/978-3-642-03741-2_3
9. Grathwohl, N.B.B., Kozen, D., Mamouras, K.: KAT + B! In: Proceedings of the Joint Meeting of the 23rd EACSL Annual Conference on Computer Science Logic (CSL) and the 29th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS 2014, pp. 44:1–44:10 (2014)
10. Huisman, M., Jacobs, B.: Java program verification via a Hoare logic with abrupt termination. In: Maibaum, T. (ed.) FASE 2000. LNCS, vol. 1783, pp. 284–303. Springer, Heidelberg (2000). doi:10.1007/3-540-46428-X_20
11. Hyland, M., Plotkin, G., Power, J.: Combining effects: sum and tensor. Theor. Comput. Sci. **357**(1), 70–99 (2006)
12. Kozen, D.: A completeness theorem for Kleene algebras and the algebra of regular events. Inf. Comput. **110**(2), 366–390 (1994)
13. Kozen, D.: Kleene algebra with tests. Trans. Programm. Lang. Syst. **19**(3), 427–443 (1997)
14. Kozen, D.: Automata on guarded strings and applications. Matématica Contemporânea **24**, 117–139 (2003)
15. Kozen, D.: Nonlocal flow of control and Kleene algebra with tests. In: Proceedings of the 23rd Annual IEEE Symposium on Logic in Computer Science (LICS 2008), pp. 105–117 (2008)
16. Kozen, D.: On the coalgebraic theory of Kleene algebra with tests. Technical report, Computing and Information Science, Cornell University, March 2008
17. Kozen, D., Mamouras, K.: Kleene algebra with equations. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014. LNCS, vol. 8573, pp. 280–292. Springer, Heidelberg (2014). doi:10.1007/978-3-662-43951-7_24
18. Kozen, D., Mamouras, K., Petrişan, D., Silva, A.: Nominal Kleene coalgebra. In: Halldórsson, M.M., Iwama, K., Kobayashi, N., Speckmann, B. (eds.) ICALP 2015. LNCS, vol. 9135, pp. 286–298. Springer, Heidelberg (2015). doi:10.1007/978-3-662-47666-6_23

19. Kozen, D., Mamouras, K., Silva, A.: Completeness and incompleteness in nominal Kleene algebra. In: Kahl, W., Winter, M., Oliveira, J.N. (eds.) RAMICS 2015. LNCS, vol. 9348, pp. 51–66. Springer, Cham (2015). doi:10.1007/978-3-319-24704-5_4
20. Kozen, D., Smith, F.: Kleene algebra with tests: completeness and decidability. In: Dalen, D., Bezem, M. (eds.) CSL 1996. LNCS, vol. 1258, pp. 244–259. Springer, Heidelberg (1997). doi:10.1007/3-540-63172-0_43
21. Kozen, D., Tiuryn, J.: On the completeness of propositional Hoare logic. Inf. Sci. **139**(3–4), 187–195 (2001)
22. Luckham, D.C., Park, D.M.R., Paterson, M.S.: On formalised computer programs. J. Comput. Syst. Sci. **4**(3), 220–249 (1970)
23. Mamouras, K.: On the Hoare theory of monadic recursion schemes. In: Proceedings of the Joint Meeting of the 23rd EACSL Annual Conference on Computer Science Logic (CSL) and the 29th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS 2014, pp. 69:1–69:10 (2014)
24. Mamouras, K.: Extensions of Kleene algebra for program verification. Ph.D. thesis, Cornell University, Ithaca, NY, August 2015
25. Mamouras, K.: Synthesis of strategies and the Hoare logic of angelic nondeterminism. In: Pitts, A. (ed.) FoSSaCS 2015. LNCS, vol. 9034, pp. 25–40. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46678-0_2
26. Mamouras, K.: The Hoare logic of deterministic and nondeterministic monadic recursion schemes. ACM Trans. Comput. Logic (TOCL) **17**(2), 13:1–13:30 (2016)
27. Mamouras, K.: Synthesis of strategies using the Hoare logic of angelic and demonic nondeterminism. Log. Methods Comput. Sci. **12**(3), 1–41 (2016)
28. Moggi, E.: Notions of computation and monads. Inf. Comput. **93**(1), 55–92 (1991)
29. von Oheimb, D.: Hoare logic for Java in Isabelle/HOL. Concurr. Comput. Pract. Exp. **13**(13), 1173–1214 (2001)
30. Plotkin, G., Power, J.: Computational effects and operations. ENTCS **73**, 149–163 (2004)
31. Plotkin, G., Pretnar, M.: Handlers of algebraic effects. In: Castagna, G. (ed.) ESOP 2009. LNCS, vol. 5502, pp. 80–94. Springer, Heidelberg (2009). doi:10.1007/978-3-642-00590-9_7
32. Pratt, V.R.: Semantical considerations on Floyd-Hoare logic. In: Proceedings of the 17th IEEE Annual Symposium on Foundations of Computer Science (FOCS 1976), pp. 109–121 (1976)
33. Saabas, A., Uustalu, T.: A compositional natural semantics and Hoare logic for low-level languages. Theor. Comput. Sci. **373**(3), 273–302 (2007)
34. Tan, G., Appel, A.W.: A compositional logic for control flow. In: Emerson, E.A., Namjoshi, K.S. (eds.) VMCAI 2006. LNCS, vol. 3855, pp. 80–94. Springer, Heidelberg (2005). doi:10.1007/11609773_6

Companions, Codensity and Causality

Damien Pous[1(✉)] and Jurriaan Rot[2,3]

[1] Univ Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP, Lyon, France
Damien.Pous@ens-lyon.fr
[2] Radboud University, Nijmegen, The Netherlands
[3] CWI, Amsterdam, The Netherlands
jrot@cs.ru.nl

Abstract. In the context of abstract coinduction in complete lattices, the notion of compatible function makes it possible to introduce enhancements of the coinduction proof principle. The largest compatible function, called the companion, subsumes most enhancements and has been proved to enjoy many good properties. Here we move to universal coalgebra, where the corresponding notion is that of a *final* distributive law. We show that when it exists the final distributive law is a monad, and that it coincides with the codensity monad of the final sequence of the given functor. On sets, we moreover characterise this codensity monad using a new abstract notion of causality. In particular, we recover the fact that on streams, the functions definable by a distributive law or GSOS specification are precisely the causal functions. Going back to enhancements of the coinductive proof principle, we finally obtain that any causal function gives rise to a valid up-to-context technique.

1 Introduction

Coinduction has been widely studied since Milner's work on CCS [26]. In concurrency theory, it is usually exploited to define behavioural equivalences or preorders on processes and to obtain powerful proof principles. Coinduction can also be used for programming languages, to define and manipulate infinite data-structures like streams or potentially infinite trees. For instance, streams can be defined using systems of differential equations [37]. In particular, pointwise addition of two streams x, y can be defined by the following equations, where x_0 and x' respectively denote the head and the tail of the stream x.

$$\begin{aligned}(x \oplus y)_0 &= x_0 + y_0\\ (x \oplus y)' &= x' \oplus y'\end{aligned} \tag{1}$$

Coinduction as a proof principle for concurrent systems can nicely be presented at the abstract level of complete lattices [30,33]: bisimilarity is the greatest

The research leading to these results has received funding from the European Research Council (FP7/2007-2013, grant agreement nr. 320571 ; and H2020, grant agreement nr. 678157); as well as from the LABEX MILYON (ANR-10-LABX-0070, ANR-11-IDEX-0007) and the project PACE (ANR-12IS02001).

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 106–123, 2017.
DOI: 10.1007/978-3-662-54458-7_7

fixpoint of a monotone function on the complete lattice of binary relations. In contrast, coinduction as a tool to manipulate infinite data-structures requires one more step to be presented abstractly: moving to universal coalgebra [15]. For instance, streams are the carrier of the final coalgebra of an endofunctor on Set, and simple systems of differential equations are just plain coalgebras.

In both cases one frequently needs enhancements of the coinduction principle [38,39]. Indeed, rather than working with plain bisimulations, which can be rather large, one often uses "bisimulations up-to", which are not proper bisimulations but are nevertheless contained in bisimilarity [1,2,10,16,24,27,40]. The situation with infinite data-structures is similar. For instance, defining the shuffle product on streams is typically done using equations of the following shape,

$$\begin{aligned} (x \otimes y)_0 &= x_0 \times y_0 \\ (x \otimes y)' &= x \otimes y' \;\oplus\; x' \otimes y \end{aligned} \tag{2}$$

which fall out of the scope of plain coinduction due to the call to pointwise addition [12,37].

Enhancements of the bisimulation proof method have been introduced by Milner from the beginning [26], and further studied by Sangiorgi [38,39] and then by the first author [30,33]. Let us recall the standard formulation of coinduction in complete lattices: by Knaster-Tarski's theorem [19,42], any monotone function b on a complete lattice admits a greatest fixpoint νb that satisfies the following *coinduction principle*:

$$\frac{x \leq y \leq b(y)}{x \leq \nu b}\ \textsc{coinduction} \tag{3}$$

In words, to prove that some point x is below the greatest fixpoint, it suffices to exhibit a point y above x which is an *invariant*, i.e., a post-fixpoint of b. Enhancements, or up-to techniques, make it possible to alleviate the second requirement: instead of working with post-fixpoints of b, one might use post-fixpoints of $b \circ f$, for carefully chosen functions f:

$$\frac{x \leq y \leq b(f(y))}{x \leq \nu b}\ \textsc{coinduction up to}\ f \tag{4}$$

Taking inspiration from the work of Hur et al. [13], the first author recently proposed to systematically use for f the largest *compatible* function [31], i.e., the largest function t such that $t \circ b \leq b \circ t$. Such a function always exists and is called the *companion*. It enjoys many good properties, the most important one possibly being that it is a closure operator: $t \circ t = t$. Parrow and Weber also characterised it extensionally in terms of the final sequence of the function b [29,31]:

$$t : x \mapsto \bigwedge_{x \leq b_\alpha} b_\alpha \qquad \text{where} \begin{cases} b_\lambda \triangleq \bigwedge_{\alpha<\lambda} b_\alpha & \text{for limit ordinals} \\ b_{\alpha+1} \triangleq b(b_\alpha) & \text{for successor ordinals} \end{cases} \tag{5}$$

In the present paper, we give a categorical account of these ideas, generalising them from complete lattices to universal coalgebra, in order to encompass important instances of coinduction such as solving systems of equations on infinite data-structures.

Let us first be more precise about our example on streams. We consider there the Set functor $BX = \mathbb{R} \times X$, whose final coalgebra is the set $\mathbb{R}^\omega$ of streams over the reals. This means that any B-coalgebra (X, f) defines a function from X to streams. Take for instance the following coalgebra over the two-elements set $2 = \{0, 1\}$: $0 \mapsto (0.3,\ 1)$, $1 \mapsto (0.7,\ 0)$. This coalgebra can be seen as a system of two equations, whose unique solution is a function from 2 to $\mathbb{R}^\omega$, i.e., two streams, where the first has value 0.3 at all even positions and 0.7 at all odd positions.

In a similar manner, one can define binary operations on streams by considering coalgebras whose carrier consists of pairs of streams. For instance, the previous system of equations characterising pointwise addition (1) is faithfully represented by the following coalgebra:

$$\begin{aligned}(\mathbb{R}^\omega)^2 &\to B((\mathbb{R}^\omega)^2)\\ (x, y) &\mapsto (x_0 + y_0,\ \ (x', y'))\end{aligned}$$

Unfortunately, as explained above, systems of equations defining operations like shuffle product (2) cannot be represented easily in this way: we would need to call pointwise addition on streams that are not yet fully defined.

To this end, one can weaken the requirement of a B-coalgebra to that of a BF-coalgebra, when there exists a distributive law $\lambda\colon FB \Rightarrow BF$ of a monad F over B [5,12]. The proof relies on the so-called generalised powerset construction [41], and this precisely amounts to using an up-to technique. Such a use of distributive laws is actually rather standard in operational semantics [5,17,43]; they properly generalise the notion of compatible function. In order to follow [31], we thus focus on the largest distributive law.

Our first contribution consists in showing that if a functor B admits a final distributive law (called the companion), then (1) this distributive law is that of a monad T over B, and (2) any BT-coalgebra has a unique morphism to the final B-coalgebra, representing a solution to the system of equations modeled by the coalgebra (Sect. 3). In complete lattices, this corresponds to the facts that the companion is a closure operator and that it can be used as an up-to technique.

Then we move to conditions under which the companion exists. We start from the *final sequence* of the functor B, which is commonly used to obtain the existence of a final coalgebra [3,4], and we show that the companion actually coincides with the *codensity monad* of this sequence, provided that this codensity monad exists and is preserved by B (Theorem 5.1). Those conditions are satisfied by all polynomial functors. This link with the final sequence of the functor makes it possible to recover Parrow and Weber's characterisation (Eq. (5)).

We can go even further for ω-continuous endofunctors on Set: the codensity monad of the final sequence can be characterised in terms of a new abstract notion of *causal algebra* (Definition 6.1). On streams, this notion coincides with

the standard notion of causality [12]: causal algebras (on streams) correspond to operations such that the n-th value of the result only depends on the n-th first values of the arguments. For instance, pointwise addition and shuffle product are causal algebras for the functor $SX = X^2$.

These two characterisations of the companion in terms of the codensity monad and in terms of causal algebras are the key theorems of the present paper. We study some of their consequences in Sect. 7.

First, given a causal algebra for a functor F, we get that any system of equations represented as a BF-coalgebra admits a unique solution. Such a technique makes it possible to define shuffle product in a streamlined way, without using distributive laws: using pointwise stream addition as a causal S-algebra, Eq. (2) can be represented by the following BS-coalgebra:

$$\begin{aligned}(\mathbb{R}^\omega)^2 &\to BS((\mathbb{R}^\omega)^2)\\ (x, y) &\mapsto (x_0 \times y_0,\ \ ((x, y')\,,\,(x', y)))\end{aligned}$$

(Intuitively, the inner pairs (x, y') and (x', y) correspond to the corecursive calls, and thus to the shuffle products $x \otimes y'$ and $x' \otimes y$; in contrast, the intermediate pair $((x, y')\,,\,(x', y))$ corresponds to a call to the causal algebra on S, i.e., in this case, pointwise addition.) In the very same way, with the functor $BX = 2 \times X^A$ for deterministic automata, we immediately obtain the semantics of non-deterministic automata and context-free grammars using simple causal algebras on formal languages (Examples 7.1 and 7.2).

Second, we obtain that algebras on the final coalgebra are causal if and only if they can be defined by a distributive law. Similar results were known to hold for streams [12] and languages [35]. Our characterisation is more abstract and less syntactic; the precise relationship between those results remains to be studied.

Third, we can combine our results with some recent work [6] where we rely on (bi)fibrations to lift distributive laws on systems (e.g., automata, LTSs) to obtain up-to techniques for coinductive predicates or relations on those systems (e.g., language equivalence, bisimilarity, divergence). Doing so, we obtain that every causal algebra gives rise to a valid up-to context technique (Sect. 7.3). For instance, bisimulation up to pointwise additions and shuffle products is a valid technique for proving stream equalities coinductively.

We conclude with an expressivity result (Sect. 8): while abstract GSOS specifications [43] seem more expressive than plain distributive laws, we show that this is actually not the case: any algebra obtained from an abstract GSOS specification can actually be defined from a plain distributive law.

2 Preliminaries

A *coalgebra* for a functor $B \colon \mathcal{C} \to \mathcal{C}$ is a pair (X, f) where X is an object in $\mathcal{C}$ and $f \colon X \to BX$ a morphism. A coalgebra homomorphism from (X, f) to (Y, g) is a $\mathcal{C}$-morphism $h \colon X \to Y$ such that $g \circ h = Fh \circ f$. A coalgebra (Z, ζ) is called *final* if it is final in the category of coalgebras , i.e., for every coalgebra (X, f) there exists a unique coalgebra morphism from (X, f) to (Z, ζ).

An *algebra* for a functor $F\colon \mathcal{D} \to \mathcal{D}$ is defined dually to a coalgebra, i.e., it is a pair (X, a) where $a\colon FX \to X$, and an algebra morphism from (X, a) to (Y, b) is a morphism $h\colon X \to Y$ such that $h \circ a = b \circ Fh$.

A *monad* is a triple (T, η, μ) where $T\colon \mathcal{C} \to \mathcal{C}$ is a functor, and $\eta\colon \mathsf{Id} \Rightarrow T$ and $\mu\colon TT \Rightarrow T$ are natural transformations called *unit* and *multiplication* respectively, such that $\mu \circ T\eta = \mathsf{id} = \mu \circ \eta T$ and $\mu \circ \mu T = \mu \circ T\mu$.

Distributive Laws. A *distributive law* of a functor $F\colon \mathcal{C} \to \mathcal{C}$ over a functor $B\colon \mathcal{C} \to \mathcal{C}$ is a natural transformation $\lambda\colon FB \Rightarrow BF$. If B has a final coalgebra (Z, ζ), then such a λ induces a unique algebra α making the following commute.

$$\begin{array}{ccccc} FZ & \xrightarrow{F\zeta} & FBZ & \xrightarrow{\lambda_Z} & BFZ \\ {\scriptstyle\alpha}\downarrow & & & & \downarrow{\scriptstyle B\alpha} \\ Z & & \xrightarrow[\zeta]{} & & BZ \end{array}$$

We call α the *algebra induced by* λ (on the final coalgebra).

Let (T, η, μ) be a monad. A distributive law of (T, η, μ) over B is a natural transformation $\lambda\colon TB \Rightarrow BT$ such that $B\eta = \lambda \circ \eta B$ and $\lambda \circ \mu B = B\mu \circ \lambda T \circ T\lambda$.

Final Sequence. Let $B\colon \mathcal{C} \to \mathcal{C}$ be an endofunctor on a complete category $\mathcal{C}$. The *final sequence* is the unique ordinal-indexed sequence defined by $B_0 = 1$ (the final object of $\mathcal{C}$), $B_{i+1} = BB_i$ and $B_j = \lim_{i<j} B_i$ for a limit ordinal j, with connecting morphisms $B_{j,i}\colon B_j \to B_i$ for all $i \leq j$, satisfying $B_{i,i} = \mathsf{id}$, $B_{j+1,i+1} = BB_{j,i}$ and if j is a limit ordinal then $(B_{j,i})_{i<j}$ is a limit cone.

The final sequence is a standard tool for constructing final coalgebras: if there exists an ordinal k such that $B_{k+1,k}$ is an isomorphism, then $B_{k+1,k}^{-1}\colon B_k \to BB_k$ is a final B-coalgebra [4, Theorem 1.3] (and dually for initial algebras [3]). In the sequel, we shall sometimes present it as a functor $\bar{B}\colon \mathsf{Ord}^{\mathsf{op}} \to \mathcal{C}$, given by $\bar{B}(i) = B_i$ and $\bar{B}(j, i) = B_{j,i}$.

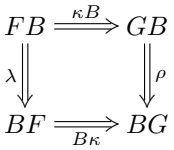
Example 2.1. Consider the functor $B\colon \mathsf{Set} \to \mathsf{Set}$ given by $BX = A \times X$, whose coalgebras are stream systems. Then $B_0 = 1$ and $B_{i+1} = A \times B_i$ for $0 < i < \omega$. Hence, for $i < \omega$, B_i is the set of all finite lists over A of length i. The limit B_ω consists of the set of all streams over A. For each i, j with $i \leq j$, the connecting map $B_{j,i}$ maps a stream (if $j = \omega$) or a list (if $j < \omega$) to the prefix of length i. The set B_ω of streams is a final B-coalgebra.

Example 2.2. For the Set functor $BX = 2 \times X^A$ whose coalgebras are deterministic automata over A, B_i is (isomorphic to) the set of languages of words over A with length below i. In particular, $B_\omega = \mathcal{P}(A^*)$ is the set of all languages, and it is a final B-coalgebra.

A functor $B\colon \mathcal{C} \to \mathcal{C}$ is called (ω)-*continuous* if it preserves limits of ω^{op}-chains. For such a functor, B_ω is the carrier of a final B-coalgebra. The functors of stream systems and automata in the above examples are both ω-continuous.

3 Properties of the Companion

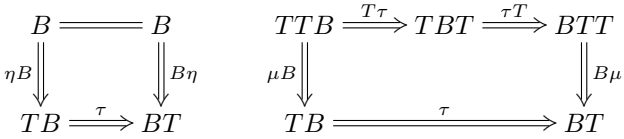
Definition 3.1. *Let $B\colon \mathcal{C} \to \mathcal{C}$ be a functor. The category $\mathsf{DL}(B)$ of distributive laws is defined as follows. An object is a pair (F, λ) where $F\colon \mathcal{C} \to \mathcal{C}$ is a functor and $\lambda\colon FB \Rightarrow BF$ is a natural transformation. A morphism from (F, λ) to (G, ρ) is a natural transformation $\kappa\colon F \Rightarrow G$ s.t. $\rho \circ \kappa B = B\kappa \circ \lambda$. The* companion *of B is the final object of $\mathsf{DL}(B)$, if it exists.*



Morphisms in $\mathsf{DL}(B)$ are a special case of *morphisms of distributive laws*, see [18,22,34,44]. In the remainder of this section, we assume that the companion of B exists, and we denote it by (T, τ). We first prove that it is a monad.

Theorem 3.1. *There are unique $\eta\colon \mathsf{Id} \Rightarrow T$ and $\mu\colon TT \Rightarrow T$ such that (T, η, μ) is a monad and $\tau\colon TB \Rightarrow BT$ is a distributive law of this monad over B.*

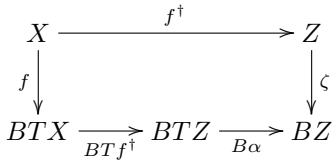
Proof. Define η and μ as the unique morphisms from id_B and $\tau T \circ T\tau$ respectively to the companion:



By definition, they satisfy the required axioms for τ to be a distributive law of monad over functor. The proof that (T, η, μ) is indeed a monad is routine, using finality of (T, τ), see the appendix [32]. □

A distributive law λ of a monad over a functor allows one to strengthen the coinduction principle obtained by finality, as observed in [5] (specifically its Corollary 4.3.6), where it is called λ-*coiteration*. This principle allows one to solve (co)recursive equations, see, e.g., loc. cit. and [14,25]. Since the companion is a distributive law of a monad (Theorem 3.1) we obtain the following.

Corollary 3.1. *Let (Z, ζ) be a final B-coalgebra. For every morphism $f\colon X \to BTX$ there is a unique morphism $f^\dagger\colon X \to Z$ such that the following commutes:*



where α is the algebra induced by the distributive law τ of the companion.

Instantiated to the complete lattice case, this is a soundness result: any invariant up to the companion (a post-fixpoint of $b \circ t$) is below the greatest fixpoint (νb).

Now assume that $\mathcal{C}$ has an initial object 0. One can define the final coalgebra and the algebra induced by the companion explicitly:

Theorem 3.2. *The B-coalgebra $(T0, \tau_0 \circ T!_{B0})$ is final, and the algebra induced on it by the companion is given by μ_0.*

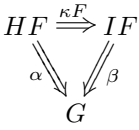
More generally, the algebra induced by any distributive law factors through the algebra μ_0 induced by the companion.

Proposition 3.1. *Let (T, η, μ) be the monad on the companion (Theorem 3.1). Let $\lambda\colon FB \Rightarrow BF$ be a distributive law, and $\alpha\colon FT0 \Rightarrow T0$ the algebra on the final coalgebra induced by it. Let $\bar{\lambda}\colon F \Rightarrow T$ be the unique natural transformation induced by finality of the companion. Then $\alpha = \mu_0 \circ \bar{\lambda}_{T0}$.*

4 The Codensity Monad

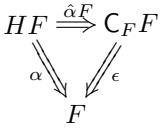
The notion of *codensity monad* is a special instance of a right Kan extension, which plays a central role in the following sections. We briefly define them here; see [20,21,28] for a comprehensive study.

Given $F\colon \mathcal{C} \to \mathcal{D}$, $G\colon \mathcal{C} \to \mathcal{E}$ be two functors. Define the category $\mathcal{K}(F,G)$ whose objects are pairs (H, α) of a functor $H\colon \mathcal{D} \to \mathcal{E}$ and a natural transformation $\alpha\colon HF \Rightarrow G$. A morphism from (H,α) to (I,β) is a natural transformation $\kappa\colon H \Rightarrow I$ such that $\beta \circ \kappa F = \alpha$.



The *right Kan extension* of G along F is a final object $(\mathsf{Ran}_F G, \epsilon)$ in $\mathcal{K}(F,G)$; the natural transformation $\epsilon\colon (\mathsf{Ran}_F G)F \Rightarrow G$ is called its *counit*. A functor $K\colon \mathcal{E} \to \mathcal{F}$ is said to *preserve* $\mathsf{Ran}_F G$ if $K \circ \mathsf{Ran}_F G$ is a right Kan extension of KG along F, with counit $K\epsilon\colon K(\mathsf{Ran}_F G)F \Rightarrow KG$.

The codensity monad is a special case, with $F = G$. Explicitly, the *codensity monad* of a functor $F\colon \mathcal{C} \to \mathcal{D}$ consists of a functor $\mathsf{C}_F\colon \mathcal{D} \to \mathcal{D}$ and a natural transformation $\epsilon\colon \mathsf{C}_F F \Rightarrow F$ s.t. for every functor $H\colon \mathcal{D} \to \mathcal{D}$ and natural transformation $\alpha\colon HF \Rightarrow F$ there is a unique $\hat{\alpha}\colon H \Rightarrow \mathsf{C}_F$ s.t. $\epsilon \circ \hat{\alpha} F = \alpha$.



As the name suggests, C_F is a monad: the unit η and the multiplication μ are the unique natural transformations such that $\epsilon \circ \eta F = \mathsf{id}$ and $\epsilon \circ \mu F = \epsilon \circ \mathsf{C}_F \epsilon$. In the sequel we will abbreviate the category $\mathcal{K}(F,F)$ as $\mathcal{K}(F)$.

Right Kan extensions can be computed pointwise as a limit, if sufficient limits exist. For an object X in $\mathcal{D}$, denote by $\Delta_X : \mathcal{C} \to \mathcal{D}$ the functor that maps every object to X. By Δ_X/F we denote the comma category, where an object is a pair (Y, f) consisting of an object Y in $\mathcal{C}$ and an arrow $f\colon X \to FY$ in $\mathcal{D}$, and an arrow from (Y,f) to (Z,g) is a map $h\colon Y \to Z$ in $\mathcal{C}$ such that $Fh \circ f = g$. There is a forgetful functor $(\Delta_X/F) \to \mathcal{C}$, which remains unnamed below.

Lemma 4.1. *Let $F\colon \mathcal{C} \to \mathcal{D}$, $G\colon \mathcal{C} \to \mathcal{E}$ be functors. If, for every object X in $\mathcal{D}$, the limit $\lim\left((\Delta_X/F) \to \mathcal{C} \xrightarrow{G} \mathcal{D}\right)$ exists, then the right Kan extension $\mathsf{Ran}_F G$ exists, and is given on an object X by that limit.*

The codensity monad of a functor F is the right Kan extension of F along itself. Hence, Lemma 4.1 gives us a way of computing the codensity monad.

The hypotheses are met in particular if $\mathcal{C}$ is essentially small (equivalent to a category with a set of objects and a set of arrows) and $\mathcal{D}$ is locally small and complete. The latter conditions hold for $\mathcal{D} = \mathsf{Set}$. In that case, we have the following concrete presentation; see, e.g., [8, Sect. 2.5] for a proof.

Lemma 4.2. *Let $F\colon \mathcal{C} \to \mathsf{Set}$ be a functor, where $\mathcal{C}$ is essentially small. The codensity monad C_F is given by $\mathsf{C}_F(X) = \{\alpha\colon (F-)^X \Rightarrow F\}$ and, for $h\colon X \to Y$, $(\mathsf{C}_F(h)(\alpha))_A\colon (FA)^Y \to FA$ is given by $f \mapsto \alpha_A(f \circ h)$. The natural transformation $\epsilon\colon \mathsf{C}_F F \Rightarrow F$ is given by $\epsilon_X(\alpha\colon F^{FX} \Rightarrow F) = \alpha_X(\mathsf{id}_{FX})$.*

5 Constructing the Companion by Codensity

It is standard in the theory of coalgebras to compute the final coalgebra of a functor B as a limit of the final sequence $\bar{B}$, see Sect. 2. In this section, we focus on the codensity monad of the final sequence, and show that it yields—under certain conditions—the companion of B.

The codensity monad of $\bar{B}$ is final in the category of natural transformations of the form $F\bar{B} \Rightarrow \bar{B}$ (see Sect. 4), whereas the companion of B is final in the category of distributive laws over B. The following lemma is a first step towards connecting companion and codensity monad.

Lemma 5.1. *For every $\lambda\colon FB \Rightarrow BF$ there exists a unique $\alpha\colon F\bar{B} \Rightarrow \bar{B}$ such that for all $i \in \mathsf{Ord}$: $\alpha_{i+1} = B\alpha_i \circ \lambda_{B_i}$. Moreover, if $B_{k+1,k}$ is an isomorphism for some k, then α_k is the algebra induced by λ on the final coalgebra.*

We turn to the main result of this section: the codensity monad of $\bar{B}$ yields the companion of B, if B preserves this codensity monad. The latter condition, as well as the concrete form of the companion computed in this manner, becomes clearer when we instantiate this result to the case where $\mathcal{C}$ is a lattice (Sect. 5.1) and the case $\mathcal{C} = \mathsf{Set}$ (Sect. 6).

Theorem 5.1. *Let $\bar{B}\colon \mathsf{Ord}^{\mathsf{op}} \to \mathcal{C}$ be the final sequence of an endofunctor B. If the codensity monad $\mathsf{C}_{\bar{B}}$ exists and B preserves it (as a right Kan extension) then there is a distributive law τ of the codensity monad $(\mathsf{C}_{\bar{B}}, \eta, \mu)$ over B such that $(\mathsf{C}_{\bar{B}}, \tau)$ is the companion of B.*

Proof (Outline). The preservation assumption means that $(B\mathsf{C}_{\bar{B}}, B\epsilon)$ is a right Kan extension of $B\bar{B}$ along $\bar{B}$. The natural transformation τ is defined, using the universal property of $B\epsilon$, as the unique $\tau\colon \mathsf{C}_{\bar{B}}B \Rightarrow B\mathsf{C}_{\bar{B}}$ such that $B\epsilon_i \circ \tau_{B_i} = \epsilon_{i+1}\colon \mathsf{C}_{\bar{B}}BB_i \Rightarrow BB_i$ for all i. See the appendix for a full proof [33]. □

The following result characterises the algebra induced on the final coalgebra by the distributive law of the companion, in terms of the counit ϵ of the codensity monad of $\bar{B}$. This plays an important role for the case $\mathcal{C} = \mathsf{Set}$ (Sect. 7).

Proposition 5.1. *Suppose B is a functor satisfying the hypotheses of Theorem 5.1. Let $(\mathsf{C}_{\bar{B}}, \epsilon)$ be the codensity monad of $\bar{B}$, with distributive law τ and monad structure $(\mathsf{C}_{\bar{B}}, \eta, \mu)$. If $B_{k+1,k}$ is an isomorphism for some k, then*

1. *$\epsilon_k\colon \mathsf{C}_{\bar{B}}B_k \to B_k$ is the algebra induced by τ on the final coalgebra;*
2. *if $\mathcal{C}$ has an initial object 0 then ϵ_k is isomorphic to μ_0.*

5.1 Codensity and the Companion of a Monotone Function

Throughout this section, let $b\colon L \to L$ be a monotone function on a complete lattice. By Theorem 5.1, the companion of a monotone function b (viewed as a functor on a poset category) is given by the right Kan extension of the final sequence $\bar{b}\colon \mathsf{Ord}^{\mathsf{op}} \to L$ along itself. Using Lemma 4.1, we obtain the characterisation of the companion given in the Introduction (5).

Theorem 5.2. *The companion t of b is given by*

$$t : x \mapsto \bigwedge_{x \leq b_i} b_i$$

Proof. By Lemma 4.1, the codensity monad $\mathsf{C}_{\bar{b}}$ can be computed by

$$\mathsf{C}_{\bar{b}}(x) = (\mathsf{Ran}_{\bar{b}}\bar{b})(x) = \bigwedge_{x \leq b_i} b_i \,,$$

a limit that exists since L is a complete lattice. We apply Theorem 5.1 to show that $\mathsf{C}_{\bar{b}}$ is the companion of b. The preservation condition of the theorem amounts to the equality $b \circ \mathsf{Ran}_{\bar{b}}\bar{b} = \mathsf{Ran}_{\bar{b}}(b \circ \bar{b})$ which, by Lemma 4.1, in turn amounts to

$$b\Big(\bigwedge_{x \leq b_i} b_i\Big) = \bigwedge_{x \leq b_i} b(b_i)$$

for all $x \in L$. The sequence $(b_i)_{i \in \mathsf{Ord}}$ is decreasing and stagnates at some ordinal ϵ; therefore, the two intersections collapse into their last terms, say b_δ and $b(b_\delta)$ (with δ the greatest ordinal such that $x \not\leq b_{\delta+1}$, or ϵ if such an ordinal does not exist). The equality follows. □

In fact, the category $\mathcal{K}(b)$ defined in Sect. 4 instantiates to the following: an object is a monotone function $f\colon L \to L$ such that $f(b_i) \leq b_i$ for all $i \in \mathsf{Ord}$, and an arrow from f to g exists iff $f \leq g$. The companion t is final in this category. This yields the following characterisation of functions below the companion.

Proposition 5.2. *Let t be the companion of b. For any monotone function f we have $f \leq t$ iff $\forall i \in \mathsf{Ord} : f(b_i) \leq b_i$.*

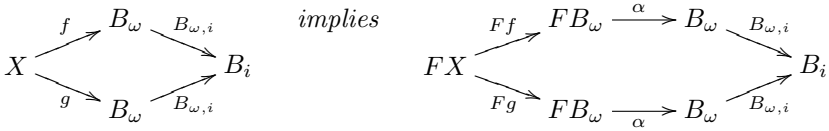
A key intuition about up-to techniques is that they should at least preserve the greatest fixpoint (i.e., up-to context is valid only when bisimilarity is a congruence). It is however well-known that this is not a sufficient condition [38,39]. The above proposition gives a stronger and better intuition: a technique should preserve all approximations of the greatest fixpoint (the elements of the final sequence) to be below the companion, and thus sound.

This intuition on complete lattices leads us to the abstract notion of causality we introduce in the following section.

6 Causality by Codensity

We focus on the codensity monad of the final sequence of an ω-continuous Set endofunctor B. For such a functor, B_ω is the carrier of a final coalgebra and Lemma 4.2 provides us with a description of the codensity monad in terms of natural transformations of the form $(\bar{B}-)^X \Rightarrow \bar{B}$. We show that such natural transformations correspond to a new abstract notion which we call *causal algebras*. Based on this correspondence and Theorem 5.1, we will get a concrete understanding of the companion of B in Sect. 7.

Definition 6.1. *Let $B, F\colon \mathsf{Set} \to \mathsf{Set}$ be functors. An algebra $\alpha\colon FB_\omega \to B_\omega$ is called (ω)-*causal *if for every set X, functions $f, g\colon X \to B_\omega$ and $i < \omega$:*



Causal algebras form a category $\mathsf{causal}(B)$*: an object is a pair $(F, \alpha\colon FB_\omega \to B_\omega)$ where α is causal, and a morphism from (F, α) to (G, β) is a natural transformation $\kappa\colon F \Rightarrow G$ such that $\beta \circ \kappa_{B_\omega} = \alpha$.*

*An (ω)-*causal function *on $|V|$ arguments is a causal algebra for the functor $(-)^V$. Equivalently, $\alpha\colon (B_\omega)^V \to B_\omega$ is causal iff for every $h, k \in (B_\omega)^V$ and every $i < \omega$: if $B_{\omega,i} \circ h = B_{\omega,i} \circ k$ then $B_{\omega,i} \circ \alpha(h) = B_{\omega,i} \circ \alpha(k)$.*

Example 6.1. Recall from Example 2.1 that, for the functor $BX = A \times X$, B_i is the set of lists of length i, and in particular B_ω is the set of streams over A. We focus first on causal *functions*. To this end, for $\sigma, \tau \in B_\omega$, we write $\sigma \equiv_i \tau$ if σ and τ are equal up to i, i.e., $\sigma(k) = \tau(k)$ for all $k < i$. It is easy to verify that a function of the form $\alpha\colon (B_\omega)^n \to B_\omega$ is causal iff for all $\sigma_1, \ldots, \sigma_n, \tau_1, \ldots, \tau_n$ and all $i < \omega$: if $\sigma_j \equiv_i \tau_j$ for all $j \leq n$ then $\alpha(\sigma_1, \ldots, \sigma_n) \equiv_i \alpha(\tau_1, \ldots, \tau_n)$.

For instance, taking $n = 2$, $\mathsf{alt}(\sigma, \tau) = (\sigma(0), \tau(1), \sigma(2), \tau(3), \ldots)$ is causal, whereas $\mathsf{even}(\sigma) = (\sigma(0), \sigma(2), \ldots)$ (with $n = 1$) is not causal. For $A = \mathbb{R}$, standard operations from the stream calculus such as pointwise stream addition, shuffle product and shuffle product are all causal.

The above notion of causal functions (with a finite set of arguments V) agrees with the standard notion of causal stream functions (e.g., [12]). Our notion of causal *algebras* generalises it from single functions to algebras for arbitrary functors. This includes polynomial functors modelling a signature. For $A = \mathbb{R}$, the algebra $\alpha\colon \mathcal{P}_\omega(B_\omega) \to B_\omega$ for the finite powerset functor $\mathcal{P}_\omega$, defined by $\alpha(S)(n) = \min\{\sigma(n) \mid \sigma \in S\}$ is a causal algebra which is not a causal function. The algebra $\beta\colon \mathcal{P}_\omega(B_\omega) \to B_\omega$ given by $\beta(S)(n) = \sum_{\sigma \in S} \sigma(n)$ is *not* causal according to Definition 6.1. Intuitively, $\beta(\{\sigma, \tau\})(i)$ depends on equality of σ and τ, since addition of real numbers is not idempotent.

Example 6.2. For the functor $BX = 2 \times X^A$, $B_\omega = \mathcal{P}(A^*)$ is the set of languages over A (Example 2.2). Given languages L and K, we write $L \equiv_i K$ if L and K

contain the same words of length below i. A function $\alpha\colon (\mathcal{P}(A^*))^n \to \mathcal{P}(A^*)$ is causal iff for all languages $L_1,\dots,L_n,K_1,\dots,K_n$: if $L_j \equiv_i K_j$ for all $j \leq n$ then $\alpha(L_1,\dots,L_n) \equiv_i \alpha(K_1,\dots,K_n)$. For instance, union, concatenation, Kleene star, and shuffle of languages are all causal. An example of a causal algebra that is not a causal function is $\alpha\colon \mathcal{P}(\mathcal{P}(A^*)) \to \mathcal{P}(A^*)$ defined by union.

The following result connects causal algebras to natural transformations of the form $F\bar{B} \Rightarrow \bar{B}$ (which, from Sect. 4, form a category $\mathcal{K}(\bar{B})$).

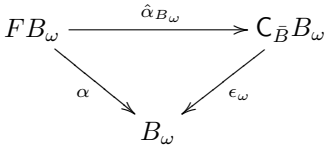
Theorem 6.1. *Let $B, F\colon \mathsf{Set} \to \mathsf{Set}$ be functors, and suppose B is ω-continuous. The category $\mathsf{causal}(B)$ of causal algebras is isomorphic to the category $\mathcal{K}(\bar{B})$. Concretely, there is a one-to-one correspondence*

$$\frac{\alpha\colon F\bar{B} \Rightarrow \bar{B}}{\alpha_\omega\colon FB_\omega \to B_\omega \text{ causal}}$$

From top to bottom, this is given by evaluation at ω. Moreover, we have $\beta \circ \kappa\bar{B} = \alpha$ iff $\beta_\omega \circ \kappa_{B_\omega} = \alpha_\omega$ for any $\alpha\colon F\bar{B} \Rightarrow \bar{B}$, $\beta\colon G\bar{B} \Rightarrow \bar{B}$ and $\kappa\colon F \Rightarrow G$.

By the above theorem, the universal property of the codensity monad amounts to the following property of causal algebras.

Corollary 6.1. *Suppose $B\colon \mathsf{Set} \to \mathsf{Set}$ is ω-continuous. Let ϵ be the counit of $\mathsf{C}_{\bar{B}}$. Then ϵ_ω is final in $\mathsf{causal}(B)$, i.e., for every causal algebra $\alpha\colon FB_\omega \to B_\omega$, there is a unique natural transformation $\hat{\alpha}\colon F \Rightarrow \mathsf{C}_{\bar{B}}$ such that $\epsilon_\omega \circ \hat{\alpha}_{B_\omega} = \alpha$.*



By Lemmas 4.2 and 6.1, we obtain the following concrete description of the codensity monad $\mathsf{C}_{\bar{B}}$ of the final sequence of a Set endofunctor B, as a *functor of causal functions*.

Theorem 6.2. *Let $B\colon \mathsf{Set} \to \mathsf{Set}$ be an ω-continuous functor. The codensity monad $\mathsf{C}_{\bar{B}}$ of the final sequence of B is given by*

$$\mathsf{C}_{\bar{B}}(X) = \{\alpha\colon B_\omega^X \to B_\omega \mid \alpha \text{ is a causal function}\},$$
$$\mathsf{C}_{\bar{B}}(h\colon X \to Y)(\alpha) = \lambda f.\, \alpha(f \circ h),$$

and, for the counit $\epsilon\colon \mathsf{C}_{\bar{B}}\bar{B} \Rightarrow \bar{B}$, we have $\epsilon_\omega(\alpha\colon B_\omega^{B\omega} \to B_\omega) = \alpha(\mathsf{id}_{B_\omega})$.

Hence, the codensity monad of the final sequence of the functor $X \mapsto A \times X$ of stream systems maps a set X to the set of all causal stream functions with $|X|$ arguments. Similarly for the functor $X \mapsto 2 \times X^A$: we obtain a functor of causal functions on languages.

7 Companion of a Set Functor

The previous sections gives us a concrete understanding of the codensity monad of the final sequence of a Set functor in terms of causal functions, and Theorem 5.1 provides us with a sufficient condition for this codensity monad to be the companion. We now focus on several applications of these results.

A rather general class of functors that satisfy the hypotheses of Theorem 5.1 is given by the *polynomial functors*. Automata, stream systems, Mealy and Moore machines, various kinds of trees, and many more are all examples of coalgebras for polynomial functors (e.g., [15]). A functor $B\colon \mathsf{Set} \to \mathsf{Set}$ is called polynomial (in a single variable) if it is isomorphic to a functor of the form

$$X \mapsto \coprod_{a \in A} X^{B_a}$$

for some A-indexed collection $(B_a)_{a\in A}$ of sets. As explained in [11, 1.18], a Set functor B is polynomial if and only if it preserves connected limits. This implies existence and preservation by B of the codensity monad of $\bar{B}$, as required by Theorem 5.1 (see the appendix for details [32]).

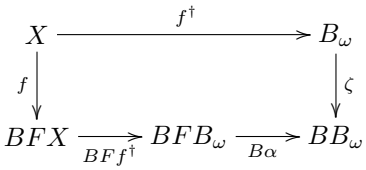
Lemma 7.1. *If $B\colon \mathsf{Set} \to \mathsf{Set}$ is polynomial, then it satisfies the hypotheses of Theorem 5.1.*

As a consequence, if B is polynomial, the functor of causal functions in Theorem 6.2 is the companion of B.

7.1 Solving Equations via Causal Algebras

As explained in the introduction, a distributive law of F over B allows one to solve systems of equations, formalised in terms of BF-coalgebras, leading to an expressive coinductive definition technique. This approach is formally supported by a solution theorem, stated for the companion in Corollary 3.1. Based on the characterisation of the companion in terms of causal algebras, we obtain a new, simplified solution theorem: it does not mention distributive laws at all, but is stated purely in terms of causal algebras.

Theorem 7.1. *Let $B\colon \mathsf{Set} \to \mathsf{Set}$ be a polynomial functor, with final coalgebra (B_ω, ζ). Let $\alpha\colon FB_\omega \to B_\omega$ be a causal algebra. For every $f\colon X \to BFX$ there is a unique $f^\dagger\colon X \to Z$ such that the following diagram commutes.*



Example 7.1. For the functor $BX = A \times X$, B_ω is the set of streams. Take $SX = X^2$ for F, and consider the coalgebra $f\colon 1 \to BS1$ with $1 = \{*\}$, defined by $* \mapsto (1, (*, *))$. Pointwise addition is a causal function on streams, modelled by an algebra on B_ω for the functor S. By Theorem 7.1 we obtain a unique solution $\sigma \in B_\omega$, satisfying $\sigma_0 = 1$ and $\sigma' = \sigma \oplus \sigma$. Similarly, the shuffle product of streams is causal, so that by applying Theorem 7.1 with that algebra and the same coalgebra f we obtain a unique stream σ satisfying $\sigma_0 = 1$, $\sigma' = \sigma \otimes \sigma$.

As explained in the Introduction, this method also allows one to define functions on streams. For instance, for the shuffle product, define a BS-coalgebra $f\colon (B_\omega)^2 \to BS(B_\omega)^2$, by $f(\sigma, \tau) = (\sigma_0 \times \tau_0, ((\sigma', \tau), (\tau, \sigma')))$. Since addition of streams is causal, by Theorem 7.1 there is a unique $f^\dagger\colon B_\omega \times B_\omega \to B_\omega$ such that $f^\dagger(\sigma, \tau)(0) = \sigma(0) \times \tau(0)$ and $(f^\dagger(\sigma, \tau))' = (f^\dagger(\sigma', \tau) \oplus f^\dagger(\sigma, \tau'))$, matching the definition given in the Introduction (2). Notice that not every function defined in this way is causal; for instance, it is easy to define even (see Example 6.1), even with the standard coinduction principle (i.e., where $F = \mathsf{Id}$ and $\alpha = \mathsf{id}$).

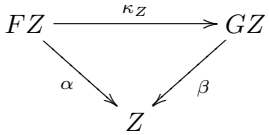
Example 7.2. Consider the functor $BX = 2 \times X^A$, whose final coalgebra consists of the set $\mathcal{P}(A^*)$ of languages. A $B\mathcal{P}$-coalgebra $f\colon X \to 2 \times (\mathcal{P}(X))^A$ is a non-deterministic automaton. Taking the causal algebra $\alpha\colon \mathcal{P}(\mathcal{P}(A^*)) \to \mathcal{P}(A^*)$ defined by union, the unique map $f^\dagger\colon X \to \mathcal{P}(A^*)$ from Theorem 7.1 is the usual language semantics of non-deterministic automata.

In [45], a context-free grammar (in Greibach normal form) is modelled as a $B\mathcal{P}^*$-coalgebra $f\colon X \to 2 \times (\mathcal{P}(X)^*)^A$, and its semantics is defined operationally by turning f into a deterministic automaton over $\mathcal{P}(X^*)$. In [36] this operational view is related to the semantics of CFGs in terms of language equations. Consider the causal algebra $\alpha\colon \mathcal{P}(\mathcal{P}(A^*)^*) \to \mathcal{P}(A^*)$ defined by union and language composition: $\alpha(S) = \bigcup_{L_1 \dots L_k \in S} L_1 L_2 \dots L_k$. By Theorem 7.1, any context-free grammar f has a unique solution in languages, which is the semantics of CFGs in the usual sense. As such, we obtain an elementary coalgebraic semantics of CFGs that does not require us to relate it to an operational semantics.

7.2 Causal Algebras and Distributive Laws

Another application of the fact that the codensity monad is the companion is that the final causal algebra in Corollary 6.1 is, by Proposition 5.1, the algebra induced by a distributive law. Hence, *any* causal algebra is "definable" by a distributive law, in the sense that it factors as a (component of a) natural transformation followed by the algebra induced by a distributive law.

More precisely, suppose $B\colon \mathsf{Set} \to \mathsf{Set}$ has a final coalgebra (Z, ζ). We say an algebra $\alpha\colon FZ \to Z$ is *definable by a distributive law over* B if there exists a distributive law $\lambda\colon GB \Rightarrow BG$ with induced algebra $\beta\colon GZ \to Z$ and a natural transformation $\kappa\colon F \Rightarrow G$ such that the following commutes:



Theorem 7.2. *Let* $B\colon \mathsf{Set} \to \mathsf{Set}$ *be polynomial. An algebra* $\alpha\colon FB_\omega \to B_\omega$ *is causal if and only if it is definable by a distributive law over* B*.*

Since the functors for stream systems and automata are polynomial, as a special case of Theorem 7.2 we obtain that a stream function, or a function on languages, is causal if and only if it is definable by a distributive law.

In [12], a similar result is shown concretely for causal stream functions, and this is extended to languages in [35]. In both cases, very specific presentations of distributive laws for the systems at hand are used to present the distributive law based on a "syntax", which however is not too clearly distinguished from the semantics: it consists of a single operation symbol for every causal function. In our case, in the proof of Theorem 7.2, we use the *companion*, which consists of the actual functions rather than a syntactic representation. Indeed, the setting of Theorem 7.2 applies more abstractly to *all* causal algebras, not just causal functions. However, it remains an intriguing question how to obtain a concrete syntactic characterisation of a distributive law for a given causal algebra.

7.3 Soundness of Up-to Techniques

The *contextual closure* of an algebra is one of the most powerful up-to techniques, which allows one to exploit algebraic structure in bisimulation proofs. In [7], it is shown that the contextual closure is sound (compatible) on any bialgebra for a distributive law. Here, we move away from distributive laws and give an elementary condition for soundness of the contextual closure on the final coalgebra: that the algebra under consideration is causal. In fact, we prove that this implies that the contextual closure lies below the companion, which not only gives soundness, but also allows to combine it with other up-to techniques.

Due to space limitations, we can not fully explain the relevant definitions, and refer to [7] for details. Bisimulations on a B-coalgebra (X, f) are the post-fixed points of a monotone function $b_f\colon \mathsf{Rel}_X \to \mathsf{Rel}_X$ on the lattice Rel_X of relations on X, defined by $b_f(R) = f^* \circ \mathsf{Rel}(B)(R)$. Here $\mathsf{Rel}(B)$ is the *relation lifting* of B, and f^* is inverse image along $f \times f$, see, e.g., [15]. Contextual closure $\mathsf{ctx}_\alpha\colon \mathsf{Rel}_X \to \mathsf{Rel}_X$ with respect to an algebra $\alpha\colon FX \to X$ is defined dually by $\mathsf{ctx}_\alpha(R) = \coprod_\alpha \circ \mathsf{Rel}(F)(R)$, where $\coprod_\alpha$ is direct image along $\alpha \times \alpha$.

Theorem 7.3. *Let* $B\colon \mathsf{Set} \to \mathsf{Set}$ *be polynomial, and* (B_ω, ζ) *a final* B*-coalgebra. Let* t_ζ *be the companion of* b_ζ*. For any causal algebra* $\alpha\colon FB_\omega \to B_\omega$*:* $\mathsf{ctx}_\alpha \leq t_\zeta$*.*

This implies that one can safely use the contextual closure for *any* causal algebra, such as union, concatenation and Kleene star of languages, or product and sum of streams. Endrullis et al. [9] prove the soundness of *causal contexts* in combination with other up-to techniques, for equality of streams. The soundness of causal algebras for streams is a special case of Theorem 7.3, but the latter provides more: being below the companion, it is possible to compose it to other such functions to obtain combined up-to techniques in a modular fashion, cf. [31].

8 Abstract GSOS

To obtain expressive specification formats, Turi and Plotkin [43] use natural transformations of the form $\lambda\colon F(B \times \mathsf{Id}) \Rightarrow BF^*$, where F^* is the free monad for F. These are the so-called *abstract GSOS specifications*. We conclude this article by showing that they are actually equally expressive as plain distributive laws of a functor F over B.

If B has a final coalgebra (Z, ζ), then any abstract GSOS specification $\lambda\colon F(B \times \mathsf{Id}) \Rightarrow BF^*$ defines an algebra $\alpha\colon FZ \to Z$ on it, which is the unique algebra making the following diagram commute.

$$\begin{array}{ccccc} FZ & \xrightarrow{F\langle\zeta,\mathsf{id}\rangle} & F(B\times\mathsf{Id})Z & \xrightarrow{\lambda_Z} & BF^*Z \\ {\scriptstyle\alpha}\downarrow & & & & \downarrow{\scriptstyle B\alpha^*} \\ Z & & \xrightarrow[\zeta]{} & & BZ \end{array}$$

Here α^* is the Eilenberg-Moore algebra for the free monad corresponding to α. Intuitively, this algebra gives the interpretation of the operations defined by λ.

Like plain distributive laws (Lemma 5.1), abstract GSOS specifications induce natural transformations of the form $F\bar{B} \Rightarrow \bar{B}$.

Lemma 8.1. *For every $\lambda\colon F(B \times \mathsf{Id}) \Rightarrow BF^*$ there is a unique $\alpha\colon F\bar{B} \Rightarrow \bar{B}$ such that for all $i \in \mathsf{Ord}$: $\alpha_{i+1} = B\alpha_i^* \circ \lambda_{B_i} \circ F\langle \mathsf{id}, B_{i+1,i}\rangle$. Moreover, if $B_{k+1,k}$ is an isomorphism for some k, then α_k is the algebra induced by λ on the final coalgebra.*

This places abstract GSOS specifications within the framework of the companion, constructed via the codensity monad of the final sequence $\bar{B}$. Whenever that construction applies (e.g., for polynomial functors), any algebra defined by an abstract GSOS is thus already definable by a plain distributive law over B.

Theorem 8.1. *Suppose $B\colon \mathcal{C} \to \mathcal{C}$ satisfies the conditions of Theorem 5.1. Every algebra induced on the final coalgebra by an abstract GSOS specification $\lambda\colon F(B \times \mathsf{Id}) \Rightarrow BF^*$ is definable by a distributive law over B (cf. Sect. 7.2).*

In this sense, abstract GSOS is no more expressive than plain distributive laws. Note, however, that this does involve moving to a different (larger) syntax.

Remark 8.1. Every abstract GSOS specification $\lambda\colon F(B \times \mathsf{Id}) \Rightarrow BF^*$ corresponds to a unique distributive law $\lambda^\dagger\colon F^*(B \times \mathsf{Id}) \Rightarrow (B \times \mathsf{Id})F^*$ of the free monad F^* over the (cofree) copointed functor $B \times \mathsf{Id}$, see [23]. The algebra induced by λ decomposes as the algebra induced by $\lambda^\dagger$ and the canonical natural transformation $F \Rightarrow F^*$. This implies that every algebra induced by an abstract GSOS is definable by a distributive law over the copointed functor $B \times \mathsf{Id}$. Theorem 8.1 strengthens this to definability by a distributive law over B.

Acknowledgments. We are grateful to Henning Basold, Filippo Bonchi, Bart Jacobs, Joshua Moerman, Daniela Petrişan, and Jan Rutten for valuable discussions and comments.

References

1. Abadi, M., Gordon, A.D.: A bisimulation method for cryptographic protocols. Nord. J. Comput. **5**(4), 267 (1998)
2. Abramsky, S.: The lazy lambda calculus. In: Research Topics in Functional Programming, pp. 65–116. Addison Wesley (1990)
3. Adámek, J.: Free algebras and automata realizations in the language of categories. Commentationes Mathematicae Universitatis Carolinae **15**(4), 589–602 (1974)
4. Barr, M.: Algebraically compact functors. J. Pure Appl. Algebra **82**(3), 211–231 (1992)
5. Bartels, F.: On generalised coinduction and probabilistic specification formats. PhD thesis, CWI, Amsterdam, April 2004
6. Bonchi, F., Petrisan, D., Pous, D., Rot, J.: Coinduction up-to in a fibrational setting. In: Proceeding CSL-LICS, pp. 20:1–20:9. ACM (2014)
7. Bonchi, F., Petrişan, D., Pous, D., Rot, J.: A general account of coinduction up-to. Acta Informatica, pp. 1–64 (2016)
8. Cordier, J.-M., Porter, T.: Shape Theory: Categorical Methods of Approximation. Mathematics and its Applications. Ellis Horwood, New York (1989). Reprinted by Dover, 2008
9. Endrullis, J., Hendriks, D., Bodin, M.: Circular coinduction in coq using bisimulation-up-to techniques. In: Blazy, S., Paulin-Mohring, C., Pichardie, D. (eds.) ITP 2013. LNCS, vol. 7998, pp. 354–369. Springer, Heidelberg (2013). doi:10.1007/978-3-642-39634-2_26
10. Fournet, C., Lévy, J.-J., Schmitt, A.: An asynchronous, distributed implementation of mobile ambients. In: Leeuwen, J., Watanabe, O., Hagiya, M., Mosses, P.D., Ito, T. (eds.) TCS 2000. LNCS, vol. 1872, pp. 348–364. Springer, Heidelberg (2000). doi:10.1007/3-540-44929-9_26
11. Gambino, N., Kock, J.: Polynomial functors and polynomial monads. In: Proceeding of Mathematical proceedings of the cambridge philosophical society, vol. 154, pp. 153–192. Cambridge University Press (2013)
12. Hansen, H.H., Kupke, C., Rutten, J.: Stream differential equations: specification formats and solution methods. CoRR, abs/1609.08367 (2016)
13. Hur, C., Neis, G., Dreyer, D., Vafeiadis, V.: The power of parameterization in coinductive proof. In: Proceeding of POPL, pp. 193–206. ACM (2013)
14. Jacobs, B.: Distributive laws for the coinductive solution of recursive equations. Inf. Comput. **204**(4), 561–587 (2006)
15. Jacobs, B.: Introduction to coalgebra. Towards mathematics of states and observations. Draft (2014)
16. Jeffrey, A., Rathke, J.: A theory of bisimulation for a fragment of concurrent ML with local names. Theor. Comput. Sci. **323**(1–3), 1–48 (2004)
17. Klin, B.: Bialgebras for structural operational semantics: an introduction. Theor. Comput. Sci. **412**(38), 5043–5069 (2011)
18. Klin, B., Nachyla, B.: Presenting morphisms of distributive laws. In: Proceeding CALCO, vol. 35. LIPIcs, pp. 190–204. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2015)

19. Knaster, B.: Un théorème sur les fonctions d'ensembles. Annales de la Socit Polonaise de Mathmatiques **6**, 133–134 (1928)
20. Lane, S.M.: Categories for the Working Mathematician. Springer, New York (1998)
21. Leinster, T.: Codensity and the ultrafilter monad. Theory and Applications of Categories **28**(13), 332–370 (2013)
22. Lenisa, M., Power, J., Watanabe, H.: Distributivity for endofunctors, pointed and co-pointed endofunctors, monads and comonads. Electron. Notes Theor. Comput. Sci. **33**, 230–260 (2000)
23. Lenisa, M., Power, J., Watanabe, H.: Category theory for operational semantics. Theor. Comput. Sci. **327**(1–2), 135–154 (2004)
24. Leroy, X.: Formal verification of a realistic compiler. Commun. ACM **52**(7), 107–115 (2009)
25. Milius, S., Moss, L.S., Schwencke, D.: Abstract GSOS rules and a modular treatment of recursive definitions. Logical Methods Comput. Sci. **9**(3) (2013)
26. Milner, R.: Communication and Concurrency. Prentice Hall (1989)
27. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes I/II. Inf. Comput. **100**(1), 1–77 (1992)
28. nLab article. Kan extension 2016. (Revision 103), http://ncatlab.org/nlab/show/Kan+extension
29. Parrow, J., Weber, T.: The largest respectful function. Logical Methods Comput. Sci. **12**(2) (2016)
30. Pous, D.: Complete lattices and up-to techniques. In: Shao, Z. (ed.) APLAS 2007. LNCS, vol. 4807, pp. 351–366. Springer, Heidelberg (2007). doi:10.1007/978-3-540-76637-7_24
31. Pous,D.: Coinduction all the way up. In: Proceeding LICS, pp. 307-316. ACM (2016)
32. Pous, D., Rot, J.: Companions, Codensity, and Causality. In: Esparza, J., Murawski, A.S. (eds.) FOSSACS 2017 LNCS, vol. 10203, pp. 106–123. Springer, Heidelberg (2017). (version with proofs), https://hal.archives-ouvertes.fr/hal-01442222
33. Pous, D., Sangiorgi, D.: Advanced Topics in Bisimulation and Coinduction, chapter about "Enhancements of the coinductive proof method". Cambridge University Press (2011)
34. Power, J., Watanabe, H.: Combining a monad and a comonad. Theor. Comput. Sci. **280**(1–2), 137–162 (2002)
35. Rot, J., Bonsangue, M.M., Rutten, J.: Proving language inclusion and equivalence by coinduction. Inf. Comput. **246**, 62–76 (2016)
36. Rot, J., Winter, J.: On language equations and grammar coalgebras for context-free languages. In: Proceeding CALCO Early Ideas (2013)
37. Rutten, J.J.M.M.: A coinductive calculus of streams. Math. Struct. Comput. Sci. **15**(1), 93–147 (2005)
38. Sangiorgi, D.: On the bisimulation proof method. Math. Struct. Comput. Sci. **8**, 447–479 (1998)
39. Sangiorgi, D., Walker, D.: The π-calculus: a theory of mobile processes. Cambridge University Press (2001)
40. Sevcík, J., Vafeiadis, V., Nardelli, F.Z., Jagannathan, S., Sewell, P.: CompCertTSO: a verified compiler for relaxed-memory concurrency. J. ACM **60**(3), 22 (2013)
41. Silva, A., Bonchi, F., Bonsangue, M., Rutten, J.: Generalizing the powerset construction, coalgebraically. In: Proceeding FSTTCS, pp. 272–283 (2010)
42. Tarski, A.: A lattice-theoretical fixpoint theorem, its applications. Pacific J. Math. **5**(2), 285–309 (1955)

43. Turi, D., Plotkin, G.D.: Towards a mathematical operational semantics. In: Proceeding LICS, pp. 280-291. IEEE (1997)
44. Watanabe, H.: Well-behaved translations between structural operational semantics. Electron. Notes Theor. Comput. Sci. **65**(1), 337–357 (2002)
45. Winter, J., Bonsangue, M.M., Rutten, J.J.M.M.: Coalgebraic characterizations of context-free languages. Logical Methods Comput. Sci. **9**(3) (2013)

Nominal Automata with Name Binding

Lutz Schröder[1(✉)], Dexter Kozen[2], Stefan Milius[1], and Thorsten Wißmann[1]

[1] Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, Germany
Lutz.Schroeder@fau.de
[2] Cornell University, Ithaca, New York, USA

Abstract. Nominal sets are a convenient setting for languages over infinite alphabets, i.e. data languages. We introduce an automaton model over nominal sets, *regular nondeterministic nominal automata (RNNA)*, which have a natural coalgebraic definition using abstraction sets to capture transitions that read a fresh letter from the input word. We prove a Kleene theorem for RNNAs w.r.t. a simple expression language that extends *nominal Kleene algebra (NKA)* with unscoped name binding, thus remedying the known failure of the expected Kleene theorem for NKA itself. We analyse RNNAs under two notions of freshness: *global* and *local*. Under global freshness, RNNAs turn out to be equivalent to session automata, and as such have a decidable inclusion problem. Under *local* freshness, RNNAs retain a decidable inclusion problem, and translate into register automata. We thus obtain decidability of inclusion for a reasonably expressive class of nondeterministic register automata, with no bound on the number of registers.

1 Introduction

Data languages are languages over infinite alphabets, regarded as modeling the communication of values from infinite data types such as nonces [23], channel names [17], process identifiers [6], URL's [2], or data values in XML documents (see [27] for a summary). There is a plethora of automata models for data languages [3,16,30], which can be classified along several axes. One line of division is between models that use explicit registers and have a finite-state description (generating infinite configuration spaces) on the one hand, and more abstract models phrased as automata over nominal sets [28] on the other hand. The latter have infinitely many states but are typically required to be *orbit-finite*, i.e. to have only finitely many states up to renaming implicitly stored letters. There are correspondences between the two styles; e.g. Bojańczyk, Klin, and Lasota's *nondeterministic orbit-finite automata (NOFA)* [5] are equivalent to Kaminski and Francez' *register automata (RAs)* [18] (originally called finite memory automata), more precisely to RAs with nondeterministic reassignment [20]. A second distinction concerns notions of freshness: *global freshness* requires that the next letter to be consumed has not been seen before, while *local freshness* postulates only that the next letter is distinct from the (boundedly many) letters currently stored in the registers.

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 124–142, 2017.
DOI: 10.1007/978-3-662-54458-7_8

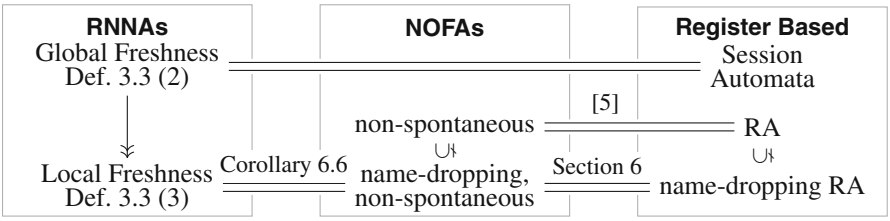


Fig. 1. Expressivity of selected data language formalisms (restricted to empty initial register assignment). FSUBAs are properly contained in name-dropping RA.

Although local freshness looks computationally more natural, nondeterministic automata models (typically more expressive than deterministic ones [21]) featuring local freshness tend to have undecidable inclusion problems. This includes RAs (unless restricted to two registers [18]) and NOFAs [5,27] as well as *variable automata* [16]. *Finite-state unification-based automata (FSUBAs)* [19] have a decidable inclusion problem but do not support freshness. Contrastingly, *session automata*, which give up local freshness in favor of global freshness, have a decidable inclusion problem [6].

Another formalism for global freshness is *nominal Kleene algebra (NKA)* [13]. It has been shown that a slight variant of the original NKA semantics satisfies one half of a Kleene theorem [21], which states that NKA expressions can be converted into a species of nondeterministic nominal automata with explicit *name binding* transitions (the exact definition of these automata being left implicit in op. cit.); the converse direction of the Kleene theorem fails even for deterministic nominal automata.

Here, we introduce *regular bar expressions (RBEs)*, which differ from NKA in making name binding dynamically scoped. RBEs are just regular expressions over an extended alphabet that includes bound letters, and hence are equivalent to the corresponding nondeterministic finite automata, which we call *bar NFAs*. We equip RBEs with two semantics capturing global and local freshness, respectively, with the latter characterized as a quotient of the former: For global freshness, we insist on bound names being instantiated with names not seen before, while in local freshness semantics, we accept also names that have been read previously but will not be used again; this is exactly the usual behaviour of α*-equivalence*, and indeed is formally defined using this notion. Under global freshness, bar NFAs are essentially equivalent to session automata.

We prove bar NFAs to be expressively equivalent to a nondeterministic nominal automaton model with name binding, *regular nondeterministic nominal automata (RNNAs)*. The states of an RNNA form an orbit-finite nominal set; RNNAs are distinguished from NOFAs by having both free and bound transitions and being finitely branching up to α-equivalence of free transitions. This is equivalent to a concise and natural definition of RNNAs as coalgebras for a functor on nominal sets (however, this coalgebraic view is not needed to understand our results). From the equivalence of bar NFAs and RNNAs we obtain

(i) a full Kleene theorem relating RNNAs and RBEs; (ii) a translation of NKA into RBEs, hence, for closed expressions, into session automata; and (iii) decidability in parametrized PSPACE of inclusion for RBEs, implying the known EXPSPACE decidability result for NKA [21].

Under local freshness, RNNAs correspond to a natural subclass of RAs (equivalently, NOFAs) defined by excluding nondeterministic reassignment and by enforcing a policy of *name dropping*, which can be phrased as "at any time, the automaton may nondeterministically lose letters from registers" – thus freeing the register but possibly getting stuck when lost names are expected to be seen later. This policy is compatible with verification problems that relate to scoping, such as 'files that have been opened need to be closed before termination' or 'currently uncommitted transactions must be either committed or explicitly aborted'. Unsurprisingly, RNNAs with local freshness semantics are strictly more expressive than FSUBAs; the relationships of the various models are summarised in Fig. 1. We show that RNNAs nevertheless retain a decidable inclusion problem under local freshness, again in parametrized PSPACE, using an algorithm that we obtain by varying the one for global freshness. This is in spite of the fact that RNNAs (a) do not impose any bound on the number of registers, and (b) allow unrestricted nondeterminism and hence express languages whose complement cannot be accepted by any RA, such as 'some letter occurs twice'.

Further Related Work. A Kleene theorem for *deterministic* nominal automata and expressions with recursion appears straightforward [21]. Kurz et al. [24] introduce regular expressions for languages over words with scoped binding, which differ technically from those used in the semantics of NKA and regular bar expressions in that they are taken only modulo α-equivalence, not the other equations of NKA concerning scope extension of binders. They satisfy a Kleene theorem for automata that incorporate a bound on the nesting depth of binding, rejecting words that exceed this depth.

Data languages are often represented as products of a classical finite alphabet and an infinite alphabet; for simplicity, we use just the set of names as the alphabet. Our unscoped name binders are, under local semantics, similar to the binders in *regular expressions with memory*, which are equivalent to unrestricted register automata [25].

Automata models for data languages, even models beyond register automata such as fresh-register automata [33] and history-register automata [15], often have decidable *emptyness* problems, and their (less expressive) deterministic restrictions then have decidable inclusion problems. Decidability of inclusion can be recovered for nondeterministic or even alternating register-based models by drastically restricting the number of registers, to at most two in the nondeterministic case [18] and at most one in the alternating case [10]. The complexity of the inclusion problem for alternating one-register automata is non-primitive recursive. *Unambiguous register automata* have a decidable inclusion problem and are closed under complement as recently shown by Colcombet et al. [8,9]. RNNAs and unambiguous RAs are incomparable: Closure under complement

implies that the language $L =$'some letter occurs twice' cannot be accepted by an unambiguous RA, as its complement cannot be accepted by any RA [4]. However, L can be accepted by an RNNA (even by an FSUBA). Failure of the reverse inclusion is due to name dropping.

Data walking automata [26] have strong navigational capabilities but no registers, and are incomparable with unrestricted RAs; we do not know how they relate to name-dropping RAs. Their inclusion problem is decidable even under nondeterminism but at least as hard as Petri net reachability, in particular not known to be elementary.

2 Preliminaries

We summarise the basics of *nominal sets*; [28] offers a comprehensive introduction.

Group Actions. Recall that an *action* of a group G on a set X is a map $G \times X \to X$, denoted by juxtaposition or infix $\cdot$, such that $\pi(\rho x) = (\pi\rho)x$ and $1x = x$ for $\pi, \rho \in G$, $x \in X$. A *G-set* is a set X equipped with an action of G. The *orbit* of $x \in X$ is the set $\{\pi x \mid \pi \in G\}$. A function $f : X \to Y$ between G-sets X, Y is *equivariant* if $f(\pi x) = \pi(fx)$ for all $\pi \in G, x \in X$. Given a G-set X, G acts on subsets $A \subseteq X$ by $\pi A = \{\pi x \mid x \in A\}$. For $A \subseteq X$ and $x \in X$, we put

$$\mathsf{fix}x = \{\pi \in G \mid \pi x = x\} \qquad \text{and} \qquad \mathsf{Fix}A = \textstyle\bigcap_{x \in A} \mathsf{fix}x.$$

Note that elements of $\mathsf{fix}\, A$ and $\mathsf{Fix}\, A$ fix A setwise and pointwise, respectively.

Nominal Sets. Fix a countably infinite set $\mathbb{A}$ of *names*, and write G for the group of finite permutations on $\mathbb{A}$. Putting $\pi a = \pi(a)$ makes $\mathbb{A}$ into a G-set. Given a G-set X and $x \in X$, a set $A \subseteq \mathbb{A}$ *supports* x if $\mathsf{Fix}\, A \subseteq \mathsf{fix}\, x$, and x *has finite support* if some finite A supports x. In this case, there is a least set $\mathsf{supp}(x)$ supporting x. We say that $a \in \mathbb{A}$ is *fresh for* x, and write $a \# x$, if $a \notin \mathsf{supp}(x)$. A *nominal set* is a G-set all whose elements have finite support. For every equivariant function f between nominal sets, we have $\mathsf{supp}(fx) \subseteq \mathsf{supp}(x)$. The function supp is equivariant, i.e. $\mathsf{supp}(\pi x) = \pi(\mathsf{supp}(x))$ for $\pi \in G$. Hence $\sharp\mathsf{supp}(x_1) = \sharp\mathsf{supp}(x_2)$ whenever x_1, x_2 are in the same orbit of a nominal set (we use $\sharp$ for cardinality). A subset $S \subseteq X$ is *finitely supported (fs)* if S has finite support with respect to the above-mentioned action of G on subsets; *equivariant* if $\pi x \in S$ for all $\pi \in G$ and $x \in S$ (which implies $\mathsf{supp}(S) = \emptyset$); and *uniformly finitely supported (ufs)* if $\bigcup_{x \in S} \mathsf{supp}(x)$ is finite [32]. We denote by $\mathcal{P}_{\mathsf{fs}}(X)$ and $\mathcal{P}_{\mathsf{ufs}}(X)$ the sets of fs and ufs subsets of a nominal set X, respectively. Any ufs set is fs but not conversely; e.g. the set $\mathbb{A}$ is fs but not ufs. Moreover, any finite subset of X is ufs but not conversely; e.g. the set of words a^n for fixed $a \in \mathbb{A}$ is ufs but not finite. A nominal set X is *orbit-finite* if the action of G on it has only finitely many orbits.

Lemma 2.1 ([12], Theorem 2.29). *If S is ufs, then* $\mathsf{supp}(S) = \bigcup_{x\in S} \mathsf{supp}(x)$.

Lemma 2.2. *Every ufs subset of an orbit-finite set X is finite.*

For a nominal set X we have the *abstraction set* [11]

$$[\mathbb{A}]X = (\mathbb{A} \times X)/{\sim}$$

where $\sim$ abstracts the notion of α-equivalence as known from calculi with name binding, such as the λ-calculus: $(a, x) \sim (b, y)$ iff $(c\,a) \cdot x = (c\,b) \cdot y$ for any fresh c. This captures the situation where x and y differ only in the concrete name given to a bound entity that is called a in x and b in y, respectively. We write $\langle a\rangle x$ for the $\sim$-equivalence class of (a, x). E.g. $\langle a\rangle\{a, d\} = \langle b\rangle\{b, d\}$ in $[\mathbb{A}]\mathcal{P}_\omega(\mathbb{A})$ provided that $d \notin \{a, b\}$.

3 Strings and Languages with Name Binding

As indicated in the introduction, we will take a simplified view of *data languages* as languages over an infinite alphabet; we will use the set $\mathbb{A}$ of names, introduced in Sect. 2, as this alphabet, so that a *data language* is just a subset $A \subseteq \mathbb{A}^*$. Much like nominal Kleene algebra (NKA) [13], our formalism will generate data words from more abstract strings that still include a form of *name binding*. Unlike in NKA, our binders will have unlimited scope to the right, a difference that is in fact immaterial at the level of strings but will be crucial at the level of regular expressions. We write a bound occurrence of $a \in \mathbb{A}$ as $|a$, and define an extended alphabet $\bar{\mathbb{A}}$ by

$$\bar{\mathbb{A}} = \mathbb{A} \cup \{\, |a \mid a \in \mathbb{A}\}.$$

Definition 3.1. A *bar string* is a word over $\bar{\mathbb{A}}$, i.e. an element of $\bar{\mathbb{A}}^*$. The set $\bar{\mathbb{A}}^*$ is made into a nominal set by the letter-wise action of G. The *free names* occurring in a bar string w are those names a that occur in w to the left of any occurrence of $|a$. A bar string is *clean* if its bound letters $|a$ are mutually distinct and distinct from all its free names. We write $\mathsf{FN}(w)$ for the set of free names of w, and say that w is *closed* if $\mathsf{FN}(w) = \emptyset$; otherwise, w is *open*. We define *α-equivalence* $\equiv_\alpha$ on bar strings as the equivalence (*not*: congruence) generated by $w|av \equiv_\alpha w|bu$ if $\langle a\rangle v = \langle b\rangle u$ in $[\mathbb{A}]\bar{\mathbb{A}}^*$ (Sect. 2). We write $[w]_\alpha$ for the α-equivalence class of w. For a bar string w, we denote by $\mathsf{ub}(w) \in \mathbb{A}^*$ (for *unbind*) the word arising from w by replacing all bound names $|a$ with the corresponding free name a.

The set $\mathsf{FN}(w)$ is invariant under α-equivalence, so we have a well-defined notion of *free names* of bar strings modulo $\equiv_\alpha$. Every bar string is α-equivalent to a clean one.

Example 3.2. We have $[ab|cab]_\alpha \neq [ab|aab]_\alpha = [ab|ccb]_\alpha \neq [ap|ccp]_\alpha$ where $\mathsf{FN}(ab|cab) = \mathsf{FN}(ab|aab) = \{a, b\}$. The bar string $ab|aab$ is not clean, and an α-equivalent clean one is $ab|ccb$.

Definition 3.3. A *literal language* is a set of bar strings, and a *bar language* is an fs set of bar strings modulo α-equivalence, i.e. an fs subset of

$$\bar{M} := \bar{\mathbb{A}}^*/\equiv_\alpha. \tag{1}$$

A literal or bar language is *closed* if all bar strings it contains are closed.

Bar languages capture *global freshness*; in fact, the operator N defined by

$$N(L) = \{\mathsf{ub}(w) \mid w \text{ clean}, [w]_\alpha \in L\} \subseteq \mathbb{A}^* \tag{2}$$

is injective on closed bar languages. Additionally, we define the *local freshness semantics* $D(L)$ of a bar language L by

$$D(L) = \{\mathsf{ub}(w) \mid [w]_\alpha \in L\} \subseteq \mathbb{A}^*. \tag{3}$$

That is, $D(L)$ is obtained by taking *all* representatives of α-equivalence classes in L and then removing bars, while N takes only clean representatives. Intuitively, D enforces local freshness by blocking α-renamings of bound names into names that have free occurrences later in the bar string. The operator D fails to be injective; e.g. (omitting notation for α-equivalence classes) $D(\{|a|b, |aa\}) = \mathbb{A}^2 = D(\{|a|b\})$. This is what we mean by our slogan that *local freshness is a quotient of global freshness*.

Remark 3.4. Again omitting α-equivalence classes, we have $D(\{|a|b\}) = \mathbb{A}^2$ because $|a|b \equiv_\alpha |a|a$. On the other hand, $D(\{|a|ba\}) = \{cdc \in \mathbb{A}^3 \mid c \neq d\}$ because $|a|ba \not\equiv_\alpha |a|aa$. We see here that since our local freshness semantics is based on α-equivalence, we can only insist on a letter d being distinct from a previously seen letter c if c will be seen again later. This resembles the process of *register allocation* in a compiler, where program variables are mapped to CPU registers (see [1, Sect. 9.7] for details): Each time the register allocation algorithm needs a register for a variable name ($|v$), any register may be (re)used whose current content is not going to be accessed later.

Remark 3.5. In *dynamic sequences* [14], there are two dynamically scoped constructs $\langle a$ and $a\rangle$ for dynamic allocation and deallocation, respectively, of a name a; in this notation, our $|a$ corresponds to $\langle aa$.

4 Regular Bar Expressions

Probably the most obvious formalism for bar languages are regular expressions, equivalently finite automata, over the extended alphabet $\bar{\mathbb{A}}$. Explicitly:

Definition 4.1. A *nondeterministic finite bar automaton*, or *bar NFA* for short, over $\mathbb{A}$ is an NFA A over $\bar{\mathbb{A}}$. We call transitions of type $q \xrightarrow{a} q$ in A *free transitions* and transitions of type $q \xrightarrow{|a} q$ *bound transitions*. The *literal language* $L_0(A)$ of A

is the language accepted by A as an NFA over $\bar{\mathbb{A}}$. The *bar language* $L_\alpha(A) \subseteq \bar{M}$ (see (1)) accepted by A is defined as

$$L_\alpha(A) = L_0(A)/{\equiv_\alpha}.$$

Generally, we denote by $L_0(q)$ the $\bar{\mathbb{A}}$-language accepted by the state q in A and by $L_\alpha(q)$ the quotient of $L_0(q)$ by α-equivalence. The *degree* $\mathsf{deg}(A)$ of A is the number of names $a \in \mathbb{A}$ that occur in transitions $q \xrightarrow{a} q'$ or $q \xrightarrow{|a} q'$ in A.

Similarly, a *regular bar expression* is a regular expression r over $\bar{\mathbb{A}}$; the *literal language* $L_0(r) \subseteq \bar{\mathbb{A}}^*$ *defined by* r is the language expressed by r as a regular expression, and the *bar language defined by* r is $L_\alpha(r) = L_0(r)/{\equiv_\alpha}$. The *degree* $\mathsf{deg}(r)$ *of* r is the number of names a occurring as either $|a$ or a in r.

Example 4.2. We have $L_\alpha(ac + |cd) = \{ac\} \cup [|cd]_\alpha$. Under local freshness semantics, this bar language contains for example ad, bd, and cd but not dd. $D(L_\alpha((a+|a)^*))$ is the same language as $D(L_\alpha(|a^*))$, even though $(a+|a)^*$ and $|a^*$ define different bar languages.

Remark 4.3. Up to the fact that we omit the finite component of the alphabet often considered in data languages, a *session automaton* [6] is essentially a bar NFA (where free names a are denoted as $a^\uparrow$, and bound names $|a$ as $a^\circledast$). It defines an $\mathbb{A}$-language and interprets bound transitions for $|a$ as binding a to some globally fresh name. In the light of the equivalence of global freshness semantics and bar language semantics in the closed case, session automata are thus essentially the same as bar NFAs; the only difference concerns the treatment of open bar strings: While session automata explicitly reject bar strings that fail to be closed (*well-formed* [6]), a bar NFA will happily accept open bar strings. Part of the motivation for this permissiveness is that we now do not need to insist on regular bar expressions to be closed; in particular, regular bar expressions are closed under subexpressions.

Example 4.4. In terms of $\mathbb{A}$-languages, bar NFAs under global freshness semantics, like session automata, can express the language "all letters are distinct" (as $|a^*$) but not the universal language $\mathbb{A}^*$ [6].

Example 4.5. The bar language $L = \{\epsilon, |ba, |ba|ab, |ba|ab|ba, |ba|ab|ba|ab \dots\}$ (omitting equivalence classes) is defined by the regular bar expression $(|ba|ab)^*(1+|ba)$ and accepted by the bar NFA A with four states s, t, u, v, where s is initial and s and u are final, and transitions $s \xrightarrow{|b} t \xrightarrow{a} u \xrightarrow{|a} v \xrightarrow{b} s$. Under global freshness, the closed bar language $|aL$ defines the language of odd-length words over $\mathbb{A}$ with identical letters in positions 0 and 2 (if any), and with every letter in an odd position being globally fresh and repeated three positions later. Under local freshness, $|aL$ defines the $\mathbb{A}$-language consisting of all odd-length words over $\mathbb{A}$ that contain the same letters in positions 0 and 2 (if any) and repeat every letter in an odd position three positions later (if any) *but no earlier*; that is, the bound names are indeed interpreted as being locally fresh. The reason for this is that, e.g., in the bar string $|a|ba|ab$, α-renaming of the bound

name $|b$ into $|a$ is blocked by the occurrence of a after $|b$; similarly, the second occurrence of $|a$ cannot be renamed into $|b$.

Example 4.6. The choice of fresh letters may restrict the branching later: The language $D(L_\alpha(|a(c + dd))) = \{ac, dc, add, cdd \mid a \in \mathbb{A}\backslash\{c, d\}\}$ contains neither bbb nor cc.

We will see in the sequel that bar NFAs and regular bar expressions are expressively equivalent to several other models, specifically

- under both semantics, to a nominal automaton model with name binding that we call *regular nondeterministic nominal automata*;
- under local freshness, to a class of nondeterministic orbit finite automata [5]; and consequently to a class of register automata.

Nominal Kleene Algebra. We recall that expressions r, s of *nominal Kleene algebra (NKA)* [13], briefly *NKA expressions*, are defined by the grammar

$$r, s ::= 0 \mid 1 \mid a \mid r + s \mid rs \mid r^* \mid \nu a.\, r \qquad (a \in \mathbb{A}).$$

Kozen et al. [21,22] give a semantics of NKA in terms of *ν-languages*. These are fs languages over words with binding, so called *ν-strings*, which are either 1 or ν-regular expressions formed using only names $a \in \mathbb{A}$, sequential composition, and name binding ν, taken modulo the equational laws of NKA [13], including α-equivalence and laws for scope extension of binding. In this semantics, a binder νa is just interpreted as itself, and all other clauses are standard. It is easy to see that the nominal set of ν-strings modulo the NKA laws is isomorphic to the universal bar language $\bar{M}$; one converts bar strings into ν-strings by replacing any occurrence of $|a$ with $\nu a.a$, with the scope of the binder extending to the end of the string. On closed expressions, ν-language semantics is equivalent to the semantics originally defined by Gabbay and Ciancia [13,22], which is given by the operator N defined in (2) (now applied also to languages containing open bar strings). Summing up, we can see NKA as another formalism for bar languages. We will see in the next section that regular bar expressions are strictly more expressive than NKA; the crucial difference is that the name binding construct νa of NKA has a static scope, while bound names $|a$ in regular bar expressions have dynamic scope.

Remark 4.7. On open expressions, the semantics of [13] and [21,22] differ as N may interpret bound names with free names appearing elsewhere in the expression; e.g. the NKA expressions $a + \nu a.\, a$ and $\nu a.\, a$ have distinct bar language semantics $\{a, |a\}$ and $\{|a\}$, respectively, which are both mapped to $\mathbb{A}$ under N. For purposes of expressivity comparisons, we will generally restrict to closed expressions as well as "closed" automata and languages in the sequel. For automata, this typically amounts to the initial register assignment being empty, and for languages to being equivariant subsets of $\bar{\mathbb{A}}^*$.

5 Regular Nondeterministic Nominal Automata

We proceed to develop a nominal automaton model that essentially introduces a notion of configuration space into the picture, and will turn out to be equivalent to bar NFAs. The deterministic restriction of our model has been considered in the context of NKA [21].

Definition 5.1. A *regular nondeterministic nominal automaton (RNNA)* is a tuple $A = (Q, \to, s, F)$ consisting of

- an orbit-finite set Q of states, with an *initial* state $s \in Q$;
- an equivariant subset $\to$ of $Q \times \bar{\mathbb{A}} \times Q$, the *transition relation*, where we write $q \xrightarrow{\alpha} q'$ for $(q, \alpha, q') \in \to$; transitions of type $q \xrightarrow{a} q'$ are called *free*, and those of type $q \xrightarrow{\restriction a} q'$ *bound*;
- an equivariant subset $F \subseteq Q$ of *final* states

such that the following conditions are satisfied:

- The relation $\to$ is *α-invariant*, i.e. closed under α-equivalence of transitions, where transitions $q \xrightarrow{\restriction a} q'$ and $p \xrightarrow{\restriction b} p'$ are *α-equivalent* if $q = p$ and $\langle a \rangle q' = \langle b \rangle p'$.
- The relation $\to$ is *finitely branching up to α-equivalence*, i.e. for each state q the sets $\{(a, q') \mid q \xrightarrow{a} q'\}$ and $\{\langle a \rangle q' \mid q \xrightarrow{\restriction a} q'\}$ are finite (equivalently ufs, by Lemma 2.2).

The *degree* $\mathsf{deg}(A) = \max\{\sharp\mathsf{supp}(q) \mid q \in Q\}$ of A is the maximum size of supports of states in A.

Remark 5.2. For readers familiar with universal coalgebra [29], we note that RNNAs have a much more compact definition in coalgebraic terms, and in fact we regard the coalgebraic definition as evidence that RNNAs are a natural class of automata; however, no familiarity with coalgebras is required to understand the results of this paper. Coalgebraically, an RNNA is simply an orbit-finite coalgebra $\gamma : Q \to FQ$ for the functor F on Nom given by

$$FX = 2 \times \mathcal{P}_{\mathsf{ufs}}(\mathbb{A} \times X) \times \mathcal{P}_{\mathsf{ufs}}([\mathbb{A}]X),$$

together with an initial state $s \in Q$. The functor F is a nondeterministic variant of the functor $KX = 2 \times X^{\mathbb{A}} \times [\mathbb{A}]X$ whose coalgebras are *deterministic nominal automata* [21]. Indeed Kozen et al. [21] show that the ν-languages, equivalently the bar languages, form the final K-coalgebra.

We proceed to define the language semantics of RNNAs.

Definition 5.3. An RNNA A, with data as above, *(literally) accepts* a bar string $w \in \bar{\mathbb{A}}^*$ if $s \xrightarrow{w} q$ for some $q \in F$, where we extend the transition notation $\xrightarrow{w}$ to bar strings in the usual way. The *literal language accepted by* A is the set $L_0(A)$ of bar strings accepted by A, and the *bar language accepted by* A is the quotient $L_\alpha(A) = L_0(A)/\equiv_\alpha$.

A key property of RNNAs is that supports of states evolve in the expected way along transitions (cf. [21, Lemma 4.6] for the deterministic case):

Lemma 5.4. *Let A be an RNNA. Then the following hold.*

1. *If $q \xrightarrow{a} q'$ in A then $\mathsf{supp}(q') \cup \{a\} \subseteq \mathsf{supp}(q)$.*
2. *If $q \xrightarrow{|a} q'$ in A then $\mathsf{supp}(q') \subseteq \mathsf{supp}(q) \cup \{a\}$.*

In fact, the properties in the lemma are clearly also sufficient for ufs branching. From Lemma 5.4, an easy induction shows that for any state q in an RNNA and any w literally accepted by A from q, we have $\mathsf{FN}(w) = \mathsf{supp}([w]_\alpha) \subseteq \mathsf{supp}(q)$. Hence:

Corollary 5.5. *Let A be an RNNA. Then $L_\alpha(A)$ is ufs; specifically, if s is the initial state of A and $w \in L_\alpha(A)$, then $\mathsf{supp}(w) \subseteq \mathsf{supp}(s)$.*

We have an evident notion of α-equivalence of paths in RNNAs, defined analogously as for bar strings. Of course, α-equivalent paths always start in the same state. The set of paths of an RNNA A is closed under α-equivalence. However, this does not in general imply that $L_0(A)$ is closed under α-equivalence; e.g. for A being

$$s() \xrightarrow{|a} t(a) \xrightarrow{|b} u(a,b) \tag{4}$$

(with a, b ranging over distinct names in $\mathbb{A}$), where $s()$ is initial and the states $u(-,-)$ are final, we have $|a|b \in L_0(A)$ but the α-equivalent $|a|a$ is not in $L_0(A)$. Crucially, closure of $L_0(A)$ under α-equivalence is nevertheless without loss of generality, as we show next.

Definition 5.6. An RNNA A is *name-dropping* if for every state q in A and every subset $N \subseteq \mathsf{supp}(q)$ there exists a state $q|_N$ in A that *restricts q to N*; that is, $\mathsf{supp}(q|_N) = N$, $q|_N$ is final if q is final, and $q|_N$ has at least the same incoming transitions as q (i.e. whenever $p \xrightarrow{\alpha} q$ then $p \xrightarrow{\alpha} q|_N$), and as many of the outgoing transitions of q as possible; i.e. $q|_N \xrightarrow{a} q'$ whenever $q \xrightarrow{a} q'$ and $\mathsf{supp}(q') \cup \{a\} \subseteq N$, and $q|_N \xrightarrow{|a} q'$ whenever $q \xrightarrow{|a} q'$ and $\mathsf{supp}(q') \subseteq N \cup \{a\}$.

The counterexample shown in (4) fails to be name-dropping, as no state restricts $q = u(a,b)$ to $N = \{b\}$. The following lemma shows that closure under α-equivalence is restored under name-dropping:

Lemma 5.7. *Let A be a name-dropping RNNA. Then $L_0(A)$ is closed under α-equivalence, i.e. $L_0(A) = \{w \mid [w]_\alpha \in L_\alpha(A)\}$.*

Finally, we can close a given RNNA under name dropping, preserving the bar language:

Lemma 5.8. *Given an RNNA of degree k with n orbits, there exists a bar language-equivalent name-dropping RNNA of degree k with at most $n2^k$ orbits.*

Proof (Sketch). From an RNNA A, construct an equivalent name-dropping RNNA with states of the form

$$q|_N := \mathsf{Fix}(N)q$$

where q is a state in A, $N \subseteq \mathsf{supp}(q)$, and $\mathsf{Fix}(N)q$ denotes the orbit of q under $\mathsf{Fix}(N)$. The final states are the $q|_N$ with q final in A, and the initial state is $s|_{\mathsf{supp}(s)}$, where s is the initial state of A. As transitions, we take

- $q|_N \xrightarrow{a} q'|_{N'}$ whenever $q \xrightarrow{a} q'$, $N' \subseteq N$, and $a \in N$, and
- $q|_N \xrightarrow{!a} q'|_{N'}$ whenever $q \xrightarrow{!b} q''$, $N'' \subseteq \mathsf{supp}(q'') \cap (N \cup \{b\})$, and $\langle a\rangle(q'|_{N'}) = \langle b\rangle(q''|_{N''})$. □

Example 5.9. Closing the RNNA from (4) under name dropping as per Lemma 5.8 yields additional states that we may denote $u(\bot, b)$ (among others), with transitions $t(a) \xrightarrow{!b} u(\bot, b)$; now, $\langle b\rangle u(\bot, b) = \langle a\rangle u(\bot, a)$, so $!a!a$ is (literally) accepted.

Equivalence to Bar NFAs. We proceed to show that RNNAs are expressively equivalent to bar NFAs by providing mutual translations. In consequence, we obtain a Kleene theorem connecting RNNAs and regular bar expressions.

Construction 5.10. We construct an RNNA $\bar{A}$ from a given bar NFA A with set Q of states, already incorporating closure under name dropping as per Lemma 5.8. For $q \in Q$, put $N_q = \mathsf{supp}(L_\alpha(q))$. The set $\bar{Q}$ of states of $\bar{A}$ consists of pairs

$$(q, \pi F_N) \qquad (q \in Q,\ N \subseteq N_q)$$

where F_N abbreviates $\mathsf{Fix}(N)$ and πF_N denotes a left coset. Left cosets for F_N can be identified with injective renamings $N \to \mathbb{A}$; intuitively, $(q, \pi F_N)$ restricts q to N and renames N according to π. (That is, we construct a configuration space, as in other translations into NOFAs [5,7]; here, we create virtual registers according to $\mathsf{supp}(L_\alpha(q))$.) We let G act on states by $\pi_1 \cdot (q, \pi_2 F_N) = (q, \pi_1\pi_2 F_N)$. The initial state of $\bar{A}$ is (s, F_{N_s}), where s is the initial state of A; a state $(q, \pi F_N)$ is final in $\bar{A}$ iff q is final in A. Free transitions in $\bar{A}$ are given by

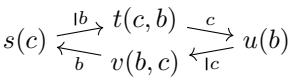
$$(q, \pi F_N) \xrightarrow{\pi(a)} (q', \pi F_{N'}) \text{ whenever } q \xrightarrow{a} q' \text{ and } N' \cup \{a\} \subseteq N$$

and bound transitions by

$$(q, \pi F_N) \xrightarrow{!a} (q', \pi' F_{N'}) \text{ whenever } q \xrightarrow{!b} q', N' \subseteq N \cup \{b\}, \langle a\rangle \pi' F_{N'} = \langle \pi(b)\rangle \pi F_{N'}.$$

Theorem 5.11. *$\bar{A}$ is a name-dropping RNNA with at most $|Q|2^{\mathsf{deg}(A)}$ orbits, $\mathsf{deg}(\bar{A}) = \mathsf{deg}(A)$, and $L_\alpha(\bar{A}) = L_\alpha(A)$.*

Example 5.12. The above construction converts the bar NFA A of Example 4.5, i.e. the expression $(|ba|ab)^*(1 + |ba)$, into an RNNA that is similar to the one appearing in the counterexample to one direction of the Kleene theorem for NKA [21] (cf. Remark 5.15): By the above description of left cosets for F_N, we annotate every state q with a list of $\sharp\mathsf{supp}(L_\alpha(q))$ entries that are either (pairwise distinct) names or $\perp$, indicating that the corresponding name from $\mathsf{supp}(L_\alpha(q))$ has been dropped. We can draw those orbits of the resulting RNNA that have the form $(q, \pi N_q)$, i.e. do not drop any names, as



for $b \neq c$, with $s(c)$, $u(b)$ final for all $b, c \in \mathbb{A}$, and $s(c)$ initial.

Additional states then arise from name dropping; e.g. for t we have additional states $t(\perp, b)$, $t(c, \perp)$, and $t(\perp, \perp)$, all with a $|b$-transition from $s(c)$. The states $t(\perp, \perp)$ and $t(\perp, b)$ have no outgoing transitions, while $t(c, \perp)$ has a c-transition to $u(\perp)$.

We next present the reverse construction, i.e. given an RNNA A we extract a bar NFA A_0 (a subautomaton of A) such that $L_\alpha(A_0) = L_\alpha(A)$.

Put $k = \mathsf{deg}(A)$. We fix a set $\mathbb{A}_0 \subseteq \mathbb{A}$ of size $\sharp\mathbb{A}_0 = k$ such that $\mathsf{supp}(s) \subseteq \mathbb{A}_0$ for the initial state s of A, and a name $* \in \mathbb{A} - \mathbb{A}_0$. The states of A_0 are those states q in A such that $\mathsf{supp}(q) \subseteq \mathbb{A}_0$. As this implies that the set Q_0 of states in A_0 is ufs, Q_0 is finite by Lemma 2.2. For $q, q' \in Q_0$, the free transitions $q \xrightarrow{a} q'$ in A_0 are the same as in A (hence $a \in \mathbb{A}_0$ by Lemma 5.4.1). The bound transitions $q \xrightarrow{|a} q'$ in A_0 are those bound transitions $q \xrightarrow{|a} q'$ in A such that $a \in \mathbb{A}_0 \cup \{*\}$. A state is final in A_0 iff it is final in A. The initial state of A_0 is $s \in Q_0$.

Theorem 5.13. *The number of states in the bar NFA A_0 is linear in the number of orbits of A and exponential in $\mathsf{deg}(A)$. Moreover, $\mathsf{deg}(A_0) \leq \mathsf{deg}(A) + 1$, and $L_\alpha(A_0) = L_\alpha(A)$.*

Combining this with Theorem 5.11, we obtain the announced equivalence result:

Corollary 5.14. *RNNAs are expressively equivalent to bar NFAs, hence to regular bar expressions.*

This amounts to a *Kleene theorem for RNNAs*. The decision procedure for inclusion (Sect. 7) will use the equivalence of bar NFAs and RNNAs, essentially running a bar NFA in synchrony with an RNNA.

Remark 5.15. It has been shown in that an NKA expression r can be translated into a nondeterministic nominal automaton whose states are the so-called *spines* of r, which amounts to one direction of a Kleene theorem [21]. One can show that the spines in fact form an RNNA, so that NKA embeds into regular bar expressions. The automata-to-NKA direction of the Kleene theorem fails

even for deterministic nominal automata, i.e. regular bar expressions are strictly more expressive than NKA. Indeed, the regular bar expression $(|ba|ab)^*(1+|ba)$ of Example 4.5 defines a language that cannot be defined in NKA because it requires unbounded nesting of name binding [21].

6 Name-Dropping Register Automata

We next relate RNNAs to two equivalent models of local freshness, nondeterministic orbit-finite automata [5] and register automata (RAs) [18]. RNNAs necessarily only capture subclasses of these models, since RAs have an undecidable inclusion problem [18]; the distinguishing condition is a version of name-dropping.

Definition 6.1. [5] A nondeterministic orbit-finite automaton (NOFA) A consists of an orbit finite set Q of states, two equivariant subsets $I, E \subseteq Q$ of *initial* and *final* states, respectively, and an equivariant transition relation $\to \subseteq Q \times \mathbb{A} \times Q$, where we write $q \xrightarrow{a} p$ for $(q,a,p) \in \to$. The $\mathbb{A}$-language $L(A) = \{w \mid A \text{ accepts } w\}$ *accepted* by A is defined in the standard way: extend the transition relation to words $w \in \mathbb{A}^*$ as usual, and then say that A *accepts* w if there exist an initial state q and a final state p such that $q \xrightarrow{w} p$. A *DOFA* is a NOFA with a deterministic transition relation.

Remark 6.2. A more succinct equivalent presentation of NOFAs is as orbit-finite coalgebras $\gamma : Q \to GQ$ for the functor

$$GX = 2 \times \mathcal{P}_{\mathsf{fs}}(\mathbb{A} \times X) \qquad (2 = \{\top, \bot\})$$

on the category Nom of nominal sets and equivariant maps, together with an equivariant subset of initial states.

More precisely speaking, NOFAs are equivalent to *RAs with nondeterministic reassignment* [5,20]. RAs are roughly described as having a finite set of registers in which names from the current word can be stored if they are *locally fresh*, i.e. not currently stored in any register; transitions are labeled with register indices k, meaning that the transition accepts the next letter if it equals the content of register k. In the equivalence with NOFAs, the names currently stored in the registers correspond to the support of states.

To enable a comparison of RNNAs with NOFAs over $\mathbb{A}$ (Sect. 5), we restrict our attention in the following discussion to RNNAs that are *closed*, i.e. whose initial state has empty support, and therefore accept equivariant $\mathbb{A}$-languages. We can convert a closed RNNA A into a NOFA $D(A)$ accepting $D(L_\alpha(A))$ by simply replacing every transition $q \xrightarrow{|a} q'$ with a transition $q \xrightarrow{a} q'$. We show that the image of this translation is a natural class of NOFAs:

Definition 6.3. A NOFA A is *non-spontaneous* if $\mathsf{supp}(s) = \emptyset$ for initial states s, and

$$\mathsf{supp}(q') \subseteq \mathsf{supp}(q) \cup \{a\} \quad \text{whenever} \quad q \xrightarrow{a} q'.$$

(In words, A is non-spontaneous if transitions $q \xrightarrow{a} q'$ in A create no new names other than a in q'.) Moreover, A is *α-invariant* if $q \xrightarrow{a} q''$ whenever $q \xrightarrow{b} q'$, $b \# q$, and $\langle a \rangle q'' = \langle b \rangle q'$ (this condition is automatic if $a \# q$). Finally, A is *name-dropping* if for each state q and each set $N \subseteq \mathsf{supp}(q)$ of names, there exists a state $q|_N$ that *restricts q to N*, i.e. $\mathsf{supp}(q|_N) = N$, $q|_N$ is final if q is final, and

- $q|_N$ has at least the same incoming transitions as q;
- whenever $q \xrightarrow{a} q'$, $a \in \mathsf{supp}(q)$, and $\mathsf{supp}(q') \cup \{a\} \subseteq N$, then $q|_N \xrightarrow{a} q'$;
- whenever $q \xrightarrow{a} q'$, $a \# q$, and $\mathsf{supp}(q') \subseteq N \cup \{a\}$, then $q|_N \xrightarrow{a} q'$.

Proposition 6.4. *A NOFA is of the form $D(B)$ for some (name-dropping) RNNA B iff it is (name-dropping and) non-spontaneous and α-invariant.*

Proposition 6.5. *For every non-spontaneous and name-dropping NOFA, there is an equivalent non-spontaneous, name-dropping, and α-invariant NOFA.*

In combination with Lemma 5.7, these facts imply

Corollary 6.6. *Under local freshness semantics, RNNAs are expressively equivalent to non-spontaneous name-dropping NOFAs.*

Corollary 6.7. *The class of languages accepted by RNNAs under local freshness semantics is closed under finite intersections.*

Proof (Sketch). Non-spontaneous name-dropping NOFAs are closed under the standard product construction. □

Remark 6.8. Every DOFA is non-spontaneous. Moreover, RAs are morally non-spontaneous according to their original definition, i.e. they can read names from the current word into the registers but cannot guess names nondeterministically [18,27]; the variant of register automata that is equivalent to NOFAs [5] in fact allows such *nondeterministic reassignment* [20]. This makes unrestricted NOFAs strictly more expressive than non-spontaneous ones [18,34]. Name-dropping restricts expressivity further, as witnessed by the language $\{ab \mid a \neq b\}$ mentioned above. In return, it buys decidability of inclusion (Sect. 7), while for non-spontaneous NOFAs even universality is undecidable [5,27]. DOFAs are incomparable to RNNAs under local freshness semantics—the language "the last letter has been seen before" is defined by the regular bar expression $(\mid b)^* \mid a (\mid b)^* a$ but not accepted by any DOFA.

Name-Dropping Register Automata and FSUBAs. In consequence of Corollary 6.6 and the equivalence between RAs and nonspontaneous NOFAs, we have that RNNAs are expressively equivalent to *name-dropping* RAs, which we just define as those RAs that map to name-dropping NOFAs under the translation given in [5]. We spend a moment on identifying a more concretely defined class of *forgetful* RAs that are easily seen to be name-dropping. We expect that forgetful RAs are as expressive as name-dropping RAs but are currently more

interested in giving a compact description of a class of name-dropping RAs to clarify expressiveness.

We use the very general definition of RAs given in [5]: An RA with n registers consists of a set C of *locations* and for each pair (c, c') of locations a *transition constraint* ϕ. *Register assignments* $w \in R := (\mathbb{A} \cup \{\bot\})^n$ determine the, possibly undefined, contents of the n registers, and *configurations* are elements of $C \times R$. Transition constraints are equivariant subsets $\phi \subseteq R \times \mathbb{A} \times R$, and $(w, a, v) \in \phi$ means that from configuration (c, w) the RA can nondeterministically go to (c', v) under input a. Transition constraints have a syntactic representation in terms of Boolean combinations of certain equations. The NOFA generated by an RA just consists of its configurations.

For $w \in R$ and $N \subseteq \mathbb{A}$ we define $w|_N \in R$ by $(w|_N)_i = w_i$ if $w_i \in N$, and $(w|_N)_i = \bot$ otherwise. An RA is *forgetful* if it generates a non-spontaneous NOFA and for every configuration (c, w) and every N, $(c, w|_N)$ restricts (c, w) to N in the sense of Definition 6.3; this property is equivalent to evident conditions on the individual transition constraints. In particular, it is satisfied if all transition constraints of the RA are conjunctions of the evident non-spontaneity restriction (letters in the poststate come from the input or the prestate) with a positive Boolean combination of the following:

- $\mathsf{cmp}_i = \{(w, a, v) \mid w_i = a\}$ (block unless register i contains the input)
- $\mathsf{store}_i = \{(w, a, v) \mid v_i \in \{\bot, a\}\}$ (store the input in register i or forget)
- $\mathsf{fresh}_i = \{(w, a, v) \mid a \neq w_i\}$ (block if register i contains the input)
- $\mathsf{keep}_{ji} = \{(w, a, v) \mid v_i \in \{\bot, w_j\}\}$ (copy register j to register i, or forget)

FSUBAs [19] can be translated into name-dropping RAs. Unlike FSUBAs, forgetful RAs do allow for freshness constraints. E.g. the language $\{aba \mid a \neq b\}$ is accepted by the forgetful RA $c_0 \xrightarrow{\mathsf{store}_1} c_1 \xrightarrow{\mathsf{fresh}_1 \wedge \mathsf{keep}_{11}} c_2 \xrightarrow{\mathsf{cmp}_1} c_3$, with c_3 final. Note how store and keep will possibly lose the content of register 1 but runs where this happens will not get past cmp_1.

7 Deciding Inclusion under Global and Local Freshness

We next show that under both global and local freshness, the inclusion problem for bar NFAs (equivalently regular bar expressions) is in ExpSpace. For global freshness, this essentially just reproves the known decidability of inclusion for session automata [6] (Remark 4.3; the complexity bound is not stated in [6] but can be extracted), while the result for local freshness appears to be new. Our algorithm differs from [6] in that it exploits name dropping; we describe it explicitly, as we will modify it for local freshness.

Theorem 7.1. *The inclusion problem for bar NFAs is in* ExpSpace*; more precisely, the inclusion $L_\alpha(A_1) \subseteq L_\alpha(A_2)$ can be checked using space polynomial in the size of A_1 and A_2 and exponential in $\mathsf{deg}(A_2)\log(\mathsf{deg}(A_1) + \mathsf{deg}(A_2) + 1)$.*

The theorem can be rephrased as saying that bar language inclusion of NFA is in parametrized polynomial space (para-PSpace) [31], the parameter being the degree.

Proof (Sketch). Let A_1, A_2 be bar NFAs with initial states s_1, s_2. We exhibit an NExpSpace procedure to check that $L_\alpha(A_1)$ is *not* a subset of $L_\alpha(A_2)$, which implies the claimed bound by Savitch's theorem. It maintains a state q of A_1 and a set Ξ of states in the name-dropping RNNA $\bar{A}_2$ generated by A_2 as described in Construction 5.10, with q initialized to s_1 and Ξ to $\{(s_2, \mathsf{id}F_{N_{s_2}})\}$. It then iterates the following:

1. Guess a transition $q \xrightarrow{\alpha} q'$ in A_1 and update q to q'.
2. Compute the set Ξ' of all states of $\bar{A}_2$ reachable from states in Ξ via α-transitions (literally, i.e. not up to α-equivalence) and update Ξ to Ξ'.

The algorithm terminates successfully and reports that $L_\alpha(A_1) \not\subseteq L_\alpha(A_2)$ if it reaches a final state q of A_1 while Ξ contains only non-final states.

Correctness of the algorithm follows from Theorem 5.11 and Lemma 5.7. For space usage, first recall that cosets πF_N can be represented as injective renamings $N \to \mathbb{A}$. Note that Ξ will only ever contain states $(q, \pi F_N)$ such that the image πN of the corresponding injective renaming is contained in the set P of names occurring literally in either A_1 or A_2. In fact, at the beginning, $\mathsf{id}N_{s_2}$ consists only of names literally occurring in A_2, and the only names that are added are those occurring in transitions guessed in Step 7, i.e. occurring literally in A_1. So states $(q, \pi F_N)$ in Ξ can be coded using partial functions $N_q \rightharpoonup P$. Since $\sharp P \leq \mathsf{deg}(A_1) + \mathsf{deg}(A_2)$, there are at most $k \cdot (\mathsf{deg}(A_1) + \mathsf{deg}(A_2) + 1)^{\mathsf{deg}(A_2)} = k \cdot 2^{\mathsf{deg}(A_2)\log(\mathsf{deg}(A_1)+\mathsf{deg}(A_2)+1)}$ such states, where k is the number of states of A_2. □

Remark 7.2. The translation from NKA expressions to bar NFAs (Remark 5.15) increases expression size exponentially but the degree only linearly. Theorem 7.1 thus implies the known ExpSpace upper bound on inclusion for NKA expressions [21].

We now adapt the inclusion algorithm to local freshness semantics. We denote by $\sqsubseteq$ the preorder (in fact: order) on $\bar{\mathbb{A}}^*$ generated by $wav \sqsubseteq w\lfloor a v$.

Lemma 7.3. *Let L_1, L_2 be bar languages accepted by RNNA. Then $D(L_1) \subseteq D(L_2)$ iff for each $[w]_\alpha \in L_1$ there exists $w' \sqsupseteq w$ such that $[w']_\alpha \in L_2$.*

Corollary 7.4. *Inclusion $D(L_\alpha(A_1)) \subseteq D(L_\alpha(A_2))$ of bar NFAs (or regular bar expressions) under local freshness semantics is in para-*PSpace*, with parameter $\mathsf{deg}(A_2)\log(\mathsf{deg}(A_1) + \mathsf{deg}(A_2) + 1)$.*

Proof. By Lemma 7.3, we can use a modification of the above algorithm where Ξ' additionally contains states of $\bar{A}_2$ reachable from states in Ξ via $\lfloor a$-transitions in case α is a free name a. □

8 Conclusions

We have studied the global and local freshness semantics of *regular nondeterministic nominal automata*, which feature explicit name-binding transitions. We have

shown that RNNAs are equivalent to session automata [6] under global freshness and to *non-spontaneous* and *name-dropping* nondeterministic orbit-finite automata (NOFAs) [5] under local freshness. Under both semantics, RNNAs are comparatively well-behaved computationally, and in particular admit inclusion checking in parameterized polynomial space. While this reproves known results on session automata under global freshness, decidability of inclusion under local freshness appears to be new. Via the equivalence between NOFAs and register automata (RAs), we in fact obtain a decidable class of RAs that allows unrestricted non-determinism and any number of registers.

Acknowledgements. We thank Charles Paperman for useful discussions, and the anonymous reviewers of an earlier version of the paper for insightful comments that led us to discover the crucial notion of name dropping. Erwin R. Catesbeiana has commented on the empty bar language.

References

1. Aho, A.V., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques, and Tools. Addison-Wesley Longman Publishing Co., Inc., Boston (1986)
2. Bielecki, M., Hidders, J., Paredaens, J., Tyszkiewicz, J., Bussche, J.: Navigating with a browser. In: Widmayer, P., Eidenbenz, S., Triguero, F., Morales, R., Conejo, R., Hennessy, M. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 764–775. Springer, Heidelberg (2002). doi:10.1007/3-540-45465-9_65
3. Bojańczyk, M.: Automata for data words and data trees. In: Rewriting Techniques and Applications, RTA 2010. LIPIcs, vol. 6, pp. 1–4. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2010)
4. Bojańczyk, M.: Computation in Sets with Atoms. http://atoms.mimuw.edu.pl/wp-content/uploads/2014/03/main.pdf
5. Bojanczyk, M., Klin, B., Lasota, S.: Automata theory in nominal sets. Log. Methods Comput. Sci. **10**, 1–44 (2014)
6. Bollig, B., Habermehl, P., Leucker, M., Monmege, B.: A robust class of data languages and an application to learning. Log. Meth. Comput. Sci. **10**, 1–23 (2014)
7. Ciancia, V., Tuosto, E.: A novel class of automata for languages on infinite alphabets. Technical report, University of Leicester, cS-09-003 (2009)
8. Colcombet, T., Puppis, G., Skrypczak, M.: Unambiguous register automata, preprint
9. Colcombet, T.: Unambiguity in automata theory. In: Shallit, J., Okhotin, A. (eds.) DCFS 2015. LNCS, vol. 9118, pp. 3–18. Springer, Cham (2015). doi:10.1007/978-3-319-19225-3_1
10. Demri, S., Lazic, R.: LTL with the freeze quantifier and register automata. ACM Trans. Comput. Log. **10**, 16:1–16:30 (2009)
11. Gabbay, M., Pitts, A.: A new approach to abstract syntax involving binders. In: Logic in Computer Science, LICS 1999, pp. 214–224. IEEE Computer Society (1999)
12. Gabbay, M.J.: Foundations of nominal techniques: logic and semantics of variables in abstract syntax. Bull. Symbolic Logic **17**(2), 161–229 (2011)
13. Gabbay, M.J., Ciancia, V.: Freshness and name-restriction in sets of traces with names. In: Hofmann, M. (ed.) FoSSaCS 2011. LNCS, vol. 6604, pp. 365–380. Springer, Heidelberg (2011). doi:10.1007/978-3-642-19805-2_25

14. Gabbay, M.J., Ghica, D.R., Petrisan, D.: Leaving the nest: nominal techniques for variables with interleaving scopes. In: Computer Science Logic, CSL 2015. LIPIcs, vol. 41, pp. 374–389. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2015)
15. Grigore, R., Tzevelekos, N.: History-register automata. Log. Meth. Comput. Sci. **12**(1), 1–32 (2016)
16. Grumberg, O., Kupferman, O., Sheinvald, S.: Variable automata over infinite alphabets. In: Dediu, A.-H., Fernau, H., Martín-Vide, C. (eds.) LATA 2010. LNCS, vol. 6031, pp. 561–572. Springer, Heidelberg (2010). doi:10.1007/978-3-642-13089-2_47
17. Hennessy, M.: A fully abstract denotational semantics for the pi-calculus. Theor. Comput. Sci. **278**, 53–89 (2002)
18. Kaminski, M., Francez, N.: Finite-memory automata. Theor. Comput. Sci. **134**, 329–363 (1994)
19. Kaminski, M., Tan, T.: Regular expressions for languages over infinite alphabets. Fund. Inform. **69**, 301–318 (2006)
20. Kaminski, M., Zeitlin, D.: Finite-memory automata with non-deterministic reassignment. Int. J. Found. Comput. Sci. **21**, 741–760 (2010)
21. Kozen, D., Mamouras, K., Petrişan, D., Silva, A.: Nominal Kleene coalgebra. In: Halldórsson, M.M., Iwama, K., Kobayashi, N., Speckmann, B. (eds.) ICALP 2015. LNCS, vol. 9135, pp. 286–298. Springer, Heidelberg (2015). doi:10.1007/978-3-662-47666-6_23
22. Kozen, D., Mamouras, K., Silva, A.: Completeness and incompleteness in nominal Kleene algebra. In: Kahl, W., Winter, M., Oliveira, J.N. (eds.) RAMICS 2015. LNCS, vol. 9348, pp. 51–66. Springer, Cham (2015). doi:10.1007/978-3-319-24704-5_4
23. Kürtz, K., Küsters, R., Wilke, T.: Selecting theories and nonce generation for recursive protocols. In: Formal methods in Security Engineering, FMSE 2007, pp. 61–70. ACM (2007)
24. Kurz, A., Suzuki, T., Tuosto, E.: On nominal regular languages with binders. In: Birkedal, L. (ed.) FoSSaCS 2012. LNCS, vol. 7213, pp. 255–269. Springer, Heidelberg (2012). doi:10.1007/978-3-642-28729-9_17
25. Libkin, L., Tan, T., Vrgoc, D.: Regular expressions for data words. J. Comput. Syst. Sci. **81**, 1278–1297 (2015)
26. Manuel, A., Muscholl, A., Puppis, G.: Walking on data words. Theor. Comput. Sys. **59**, 180–208 (2016)
27. Neven, F., Schwentick, T., Vianu, V.: Finite state machines for strings over infinite alphabets. ACM Trans. Comput. Log. **5**, 403–435 (2004)
28. Pitts, A.: Nominal Sets: Names and Symmetry in Computer Science. Cambridge University Press, Cambridge (2013)
29. Rutten, J.: Universal coalgebra: a theory of systems. Theor. Comput. Sci. **249**(1), 3–80 (2000)
30. 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). doi:10.1007/11874683_3
31. Stockhusen, C., Tantau, T.: Completeness results for parameterized space classes. In: Gutin, G., Szeider, S. (eds.) IPEC 2013. LNCS, vol. 8246, pp. 335–347. Springer, Cham (2013). doi:10.1007/978-3-319-03898-8_28

32. Turner, D., Winskel, G.: Nominal domain theory for concurrency. In: Grädel, E., Kahle, R. (eds.) CSL 2009. LNCS, vol. 5771, pp. 546–560. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04027-6_39
33. Tzevelekos, N.: Fresh-register automata. In: Principles of Programming Languages, POPL 2011, pp. 295–306. ACM (2011)
34. Wysocki, T.: Alternating register automata on finite words. Master's thesis, University of Warsaw (2013). (In Polish)

Games and Automata

On the Existence of Weak Subgame Perfect Equilibria

Véronique Bruyère[1(✉)], Stéphane Le Roux[2], Arno Pauly[2], and Jean-François Raskin[2]

[1] Département d'informatique, Université de Mons (UMONS), Mons, Belgium
veronique.bruyere@umons.ac.be
[2] Département d'informatique, Université Libre de Bruxelles (ULB), Brussels, Belgium

Abstract. We study multi-player turn-based games played on a directed graph, where the number of players and vertices can be infinite. An outcome is assigned to every play of the game. Each player has a preference relation on the set of outcomes which allows him to compare plays. We focus on the recently introduced notion of weak subgame perfect equilibrium (weak SPE), a variant of the classical notion of SPE, where players who deviate can only use strategies deviating from their initial strategy in a finite number of histories. We give general conditions on the structure of the game graph and the preference relations of the players that guarantee the existence of a weak SPE, which moreover is finite-memory.

1 Introduction

Games played on graphs have a large number of applications in theoretical computer science. One particularly important application is *reactive synthesis* [21], i.e. the design of a controller that guarantees a good behavior of a reactive system evolving in a possibly hostile environment. One classical model proposed for the synthesis problem is the notion of *two-player zero-sum game played on a graph*. One player is the reactive system and the other one is the environment; the vertices of the graph model their possible states and the edges model their possible actions. Interactions between the players generate an infinite play in the graph which model behaviors of the system within its environment. As one cannot assume cooperation of the environment, the objectives of the two players are considered to be opposite. Constructing a controller for the system then means devising a *winning strategy* for the player modeling it. Reality is often more subtle and the environment is usually not fully adversarial as it has its own objective, meaning that the game should be non zero-sum. Moreover instead of two players, we could consider the more general situation of several players modeling different interacting systems/environments each of them with its own objective.

S. Le Roux, A. Pauly, J.-F. Raskin—Supported by ERC Starting Grant (279499: inVEST).

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 145–161, 2017.
DOI: 10.1007/978-3-662-54458-7_9

The concept of *Nash equilibrium* (NE) [20] is central to the study of *multi-player non zero-sum games*. A strategy profile is an NE if no player has an incentive to deviate unilaterally from his strategy, i.e., he cannot strictly improve the outcome of the strategy profile by changing his strategy only. However in the context of games played on graphs, which are sequential by nature, it is well-known that NEs present a serious drawback: they allow for *non-credible threats* that rational players should not carry out [23]. Thus the notion of NE has been strengthened into the notion of *subgame perfect equilibrium* (SPE) [24]: a strategy profile is an SPE if it is an NE in each subgame of the original game. This notion behaves better for sequential games and excludes non-credible threats.

Variants of SPE, *weak SPE* and *very weak SPE*, have been recently introduced in [5]. While an SPE must be resistant to any unilateral deviation of one player, a weak (resp. very weak) SPE must be resistant to such deviations where the deviating strategy differs from the original one on a *finite number* of histories only (resp. on the *initial vertex* only). The latter class of deviating strategies is a well-known notion that for instance appears in the proof of Kuhn's theorem [16] with the one-step deviation property. Weak SPEs and very weak SPEs are equivalent, but there are games for which there exists a weak SPE but no SPE [5,26]. The notion of weak SPE is important for several reasons (more details are given in the related work discussed below). First, for the large class of games with upper-semicontinuous payoff functions and for games played on finite trees, the notions of SPE and weak SPE are equivalent. Second, it is a central technical ingredient used to reason on SPEs as shown in [5,12]. Third, being immune to strategies that finitely deviate from the initial strategy profile may be sufficient from a practical point of view. Indeed ruling out infinite deviations can be achieved by letting a meta-agent punish every one-shot deviation with a (low) fixed probability. A player using an infinitely-deviating strategy will thus be punished by the meta-agent with probability one. Protocols like BitTorrent use similar ideas: every deviant user is temporarily denied suitable bandwidth (see Chapter *Bandwidth Trading as Incentive* in [25] for details).

In this paper, we provide the following contributions. First, we identify *general conditions* to guarantee the existence of a weak SPE (Theorem 1). The result identifies a large class of multi-player non zero-sum games such that an outcome is assigned to every play of the game and each player has a preference relation on the set of play outcomes which allows him to compare plays. This class covers game graphs that may have infinitely many vertices and infinitely many players. Notice that such models are relevant for systems where the players can join or leave the game dynamically, and the number of players is finite yet unbounded overtime: the users in the Internet are a typical example since there is no (clear) bound on the number of possible users. The proof of our result relies on transfinite induction and additionally provides a weak SPE using finite-memory strategies for all players. Second, starting from this general existence result, we prove the existence of a weak SPE:

– for games with a *finite* number of outcomes (Theorem 2);
– for games with a *finite* underlying graph and a *prefix-independent* outcome function (Theorem 4).

Additionally, in the second result, we identify conditions on the players' outcome preferences that guarantee the existence of a weak SPE composed of *uniform memoryless* strategies only (Theorem 5).

Related work. The concept of SPE has been first introduced and studied by the game theory community. Kuhn proves in [16] the existence of SPEs in games played on finite trees. This result has been generalized in several ways. Games with a continuous real-valued outcome function and a finitely branching tree always have an SPE [19] (the case with finitely many players is first established in [14]). In [12] (resp. [22]), the authors prove that there always exists an SPE for games with a finite number of players and with a real-valued outcome function that is upper-semicontinuous (resp. lower-semicontinuous) and of finite range. The result of [22] is extended to an infinite number of players in [13]. In [19], it is proved using Borel determinacy that all two-player games with antagonistic preferences over finitely many outcomes and a Borel-measurable outcome function have an SPE. In [18], Le Roux shows that all games where the preferences over finitely many outcomes are free of some "bad pattern" and the outcome function is Δ_2^0 measurable (a low level in the Borel hierarchy) have an SPE.

In part of the former work, the equivalence between SPEs and very weak SPEs is implicitly used as a proof technique: in a finite setting in [16], continuous setting in [14], and lower-semicontinuous setting in [12]. In the latter reference, the authors implicitly prove that all games with a finite range real-valued outcome function have a weak SPE (which is an SPE when the outcome function is additionally lower-semicontinuous). Inspired by this result and its proof, we here generalize it to an infinite number of players using a simpler proof technique: our algorithm discards outcomes instead of discarding plays.

The concept of SPE and other solution concepts for multi-player non zero-sum games have been recently studied by the theoretical computer community, see [2] for a survey. In [27], the existence of SPEs (and thus weak SPEs) is proved for games with a finite number of players and Borel Boolean objectives. We here generalize the existence of weak SPEs to games with infinitely many players. In [5], weak SPEs are introduced as a technical tool for showing the existence of SPEs in quantitative reachability games played on finite weighted graphs. An algorithm is also provided for the construction of a (finite-memory) weak SPE that appears to be an SPE for this particular class of games. We here give several existence results that are orthogonal to the results of [5] as they are concerned with possibly infinite graphs or prefix-independent outcome functions.

Other refinements of NE are studied. Let us mention the secure equilibria for two players first introduced in [7] and then used for reactive synthesis in [10]. These equilibria are generalized to multiple players in [11] or to quantitative objectives in [6], see also a variant called Doomsday equilibrium in [8]. Like NEs, they are subject to possible non-credible threats. Other refinements of NE

are provided by the notion of admissible strategy introduced in [1], with computational aspects studied in [4], and potential for synthesis studied in [3]. Note that these notions are immune, as (weak) SPEs, of non-credible threats. Finally, in [17], the authors introduce the notion of cooperative and non-cooperative rational synthesis as a general framework where rationality can be specified by either NE, or SPE, or the notion of dominating strategies. In all cases except [6] and [11], the proposed solution concepts are not guaranteed to exist, hence results concern mostly algorithmic techniques to decide their existence and not general conditions for existence as in this paper.

2 Preliminaries

In this section, we consider multi-player turn-based games such that an outcome is assigned to every play. Each player has a preference relation on the set of play outcomes which allows him to compare plays.

Games. A *game* is a tuple $G = (\Pi, V, (V_i)_{i\in\Pi}, E, O, \mu, (\prec_i)_{i\in\Pi})$ where *(i)* Π is a set of players, *(ii)* V is a set of vertices and $E \subseteq V \times V$ is a set of edges, such that w.l.o.g. each vertex has at least one outgoing edge, *(iii)* $(V_i)_{i\in\Pi}$ is a partition of V such that V_i is the set of vertices controlled by player $i \in \Pi$, *(iv)* O is a set of outcomes and $\mu : V^\omega \to O$ is an outcome function, and *(v)* $\prec_i \subseteq O \times O$ is a preference relation for player $i \in \Pi$. In this definition the underlying graph (V, E) can be infinite (that is, of arbitrarily cardinality), as well as the set Π of players and the set O of outcomes.

A *play* of G is an infinite (countable) sequence $\rho = \rho_0\rho_1 \ldots \in V^\omega$ of vertices such that $(\rho_i, \rho_{i+1}) \in E$ for all $i \in \mathbb{N}$. *Histories* of G are finite sequences $h = h_0 \ldots h_n \in V^+$ defined in the same way. We often use notation hv to mention the last vertex $v \in V$ of the history. Usually histories are non empty, but in specific situations it will be useful to consider the empty history ϵ. The set of plays is denoted by $Plays$ and the set of histories (ending with a vertex in V_i) by $Hist$ (resp. by $Hist_i$).[1] A *prefix* (resp. *suffix*) of a play $\rho = \rho_0\rho_1 \ldots$ is a finite sequence $\rho_{\leq n} = \rho_0 \ldots \rho_n$ (resp. infinite sequence $\rho_{\geq n} = \rho_n\rho_{n+1} \ldots$). We use notation $h < \rho$ when a history h is prefix of a play ρ. When an initial vertex $v_0 \in V$ is fixed, we call (G, v_0) an *initialized* game. In this case, plays and histories are supposed to start in v_0, and we use notations $Plays(v_0)$ and $Hist(v_0)$. In this article, we often *unravel* the graph of the game (G, v_0) from the initial vertex v_0, which yields an infinite tree rooted at v_0.

The outcome function μ assigns an outcome $\mu(\rho) \in O$ to each play $\rho \in V^\omega$. It is *prefix-independent* if $\mu(h\rho) = \mu(\rho)$ for all histories h and play ρ. A *preference* relation $\prec_i \subseteq O \times O$ is an irreflexive and transitive binary relation. It allows for player i to compare two plays $\rho, \rho' \in V^\omega$ with respect to their outcome: $\mu(\rho) \prec_i \mu(\rho')$ means that player i prefers ρ' to ρ. In this paper we restrict to *linear* preferences. (It is w.l.o.g. since the preference properties that we use are preserved by linear extension). We write $o \preceq_i o'$ when $o \prec_i o'$ or $o = o'$; notice

[1] Indexing $Plays_G$ or $Hist_G$ with G allows to recall the related game G.

that $o \not\prec_i o'$ if and only if $o' \preceq_i o$. We sometimes use notation $\prec_v$ instead of $\prec_i$ when vertex $v \in V_i$ is controlled by player i.

Example 1. Let us mention some classical classes of games where the set of outcomes O is a subset of $(\mathbb{R} \cup \{+\infty, -\infty\})^{\Pi}$, and for all player $i \in \Pi$, $\prec_i$ is the usual ordering $<$ on $\mathbb{R} \cup \{+\infty, -\infty\}$ on the outcome i-th components. In other words, each player i has a real-valued payoff function $\mu_i : Plays \to \mathbb{R} \cup \{+\infty, -\infty\}$. The outcome function of the game is then equal to $\mu = (\mu_i)_{i \in \Pi}$, and for all $i \in \Pi$, $\mu(\rho) \prec_i \mu(\rho')$ whenever $\mu_i(\rho) < \mu_i(\rho')$.

Games with *Boolean* objectives are such that $\mu_i : Plays \to \{0, 1\}$ where 1 (resp. 0) means that the play is won (resp. lost) by player i. Classical objectives are Borel objectives including ω-regular objectives, like reachability, Büchi, parity, also [15]. Prefix-independence of μ_i holds in the case of Büchi and parity objectives, but not for reachability objective.

We have *quantitative* objectives when $\mu_i : Plays \to \mathbb{R} \cup \{+\infty, -\infty\}$ replaces $\mu_i : Plays \to \{0, 1\}$. Usually, such a μ_i is defined from a weight function $w_i : E \to \mathbb{R}$ that assigns a weight to each edge. Classical examples of μ_i are *limsup* and *mean-payoff* functions [9][2]: *(i) limsup*: $\mu_i(\rho) = \limsup_{k \to \infty} w_i(\rho_k, \rho_{k+1})$, *(ii) mean-payoff*: $\mu_i(\rho) = \limsup_{n \to \infty} \sum_{k=0}^{n} \frac{w_i(\rho_k, \rho_{k+1})}{n}$.

Strategies. Let (G, v_0) be an initialized game. A *strategy* σ for player i in (G, v_0) is a function $\sigma : Hist_i(v_0) \to V$ assigning to each history $hv \in Hist_i(v_0)$ a vertex $v' = \sigma(hv)$ such that $(v, v') \in E$. A strategy σ of player i is *positional* if it only depends on the last vertex of the history, i.e. $\sigma(hv) = \sigma(v)$ for all $hv \in Hist_i(v_0)$. It is a *finite-memory* strategy if $\sigma(hv)$ only needs finite memory of the history hv (recorded by a Moore machine[3] with a finite number of memory states). These definitions of (positional, finite-memory) strategy are given for an initialized game (G, v_0). We call *uniform* every positional strategy σ of player i defined for all $hv \in Hist_i$ (instead of $Hist_i(v_0)$), that is, when σ is a positional strategy in all initialized games (G, v), $v \in V$.

A play ρ is *consistent* with a strategy σ of player i if $\rho_{n+1} = \sigma(\rho_{\leq n})$ for all n such that $\rho_n \in V_i$. A *strategy profile* is a tuple $\bar{\sigma} = (\sigma_i)_{i \in \Pi}$ of strategies, where each σ_i is a strategy of player i. It is called *positional* (resp. *finite-memory with memory size bounded by c*, *uniform*) if all σ_i, $i \in \Pi$, are positional (resp. finite-memory with memory size bounded by c, uniform). Given an initial vertex v_0, such a strategy profile induces a unique play of (G, v_0) consistent with all the strategies, denoted by $\langle \bar{\sigma} \rangle_{v_0}$. We say that $\bar{\sigma}$ has outcome $\mu(\langle \bar{\sigma} \rangle_{v_0})$.

Let $\bar{\sigma}$ be a strategy profile. When all players stick to their own strategy except player i that shifts from σ_i to σ'_i, we denote by $(\sigma'_i, \bar{\sigma}_{-i})$ the derived strategy profile, and by $\langle \sigma'_i, \bar{\sigma}_{-i} \rangle_{v_0}$ the induced play in (G, v_0). We say that σ'_i is a *deviating* strategy from σ_i. When σ_i and σ'_i only differ on a finite number of histories (resp. on v_0), we say that σ'_i is a *finitely-deviating* (resp. *one-shot deviating*) strategy from σ_i. One-shot deviating strategies is a well-known notion

[2] The limit inferior can be used instead of the limit superior.

[3] Moore machines are usually defined for finite sets V. We here allow infinite sets V.

that for instance appears in the proof of Kuhn's theorem [16] with the one-step deviation property. Finitely-deviating strategies have been introduced in [5].

Variants of subgame perfect equilibria. Let us first recall the classical notion of Nash equilibrium (NE). Informally, a strategy profile $\bar{\sigma}$ in an initialized game (G, v_0) is an NE if no player has an incentive to deviate (with respect to his preference relation), if the other players stick to their strategies.

Definition 1. *Given an initialized game (G, v_0), a strategy profile $\bar{\sigma} = (\sigma_i)_{i \in \Pi}$ of (G, v_0) is a* Nash equilibrium *if for all players $i \in \Pi$, for all strategies σ'_i of player i, we have $\mu(\langle \bar{\sigma} \rangle_{v_0}) \not\prec_i \mu(\langle \sigma'_i, \bar{\sigma}_{-i} \rangle_{v_0})$.*

When $\mu(\langle \bar{\sigma} \rangle_{v_0}) \prec_i \mu(\langle \sigma'_i, \bar{\sigma}_{-i} \rangle_{v_0})$, we say that σ'_i is a *profitable deviation* for player i w.r.t. $\bar{\sigma}$.

The notion of subgame perfect equilibrium (SPE) is a refinement of NE. To define it, we introduce the following concepts. Given a game $G = (\Pi, V, (V_i)_{i \in \Pi}, E, \mu, (\prec_i)_{i \in \Pi})$ and a history $h \in Hist$, we denote by $G_{|h}$ the game $(\Pi, V, (V_i)_{i \in \Pi}, E, \mu_{|h}, (\prec_i)_{i \in \Pi})$ where $\mu_{|h}(\rho) = \mu(h\rho)$ for all plays of $G_{|h}$[4], and we say that $G_{|h}$ is a *subgame* of G. Given an initialized game (G, v_0) and a history $hv \in Hist(v_0)$, the initialized game $(G_{|h}, v)$ is called the subgame of (G, v_0) with history hv. In particular (G, v_0) is a subgame of itself with history hv_0 such that $h = \epsilon$. Given a strategy σ of player i in (G, v_0), the strategy $\sigma_{|h}$ in $(G_{|h}, v)$ is defined as $\sigma_{|h}(h') = \sigma(hh')$ for all $h' \in Hist_i(v)$. Given a strategy profile $\bar{\sigma}$ in (G, v_0), we use notation $\bar{\sigma}_{|h}$ for $(\sigma_{i|h})_{i \in \Pi}$, and $\langle \bar{\sigma}_{|h} \rangle_v$ is the induced play in the subgame $(G_{|h}, v)$.

Now a strategy profile is an SPE in an initialized game if it induces an NE in each of its subgames. Two variants of SPE, called weak SPE and very weak SPE, are proposed in [5] such that no player has an incentive to deviate in any subgame using finitely deviating strategies and one-shot deviating strategies respectively (instead of any deviating strategy).

Definition 2. *Given an initialized game (G, v_0), a strategy profile $\bar{\sigma}$ of (G, v_0) is a* (weak, very weak resp.) subgame perfect equilibrium *if for all histories $hv \in Hist(v_0)$, for all players $i \in \Pi$, for all (finitely, one-shot resp.) deviating strategies σ'_i from $\sigma_{i|h}$ of player i in the subgame $(G_{|h}, v)$, we have $\mu(\langle \bar{\sigma}_{|h} \rangle_v) \not\prec_i \mu(\langle \sigma'_i, \bar{\sigma}_{-i|h} \rangle_v)$.*

Every SPE is a weak SPE, and every weak SPE is a very weak SPE. The next proposition states that weak SPE and very weak SPE are equivalent notions, but this is not true for SPE and weak SPE (see also Example 2 below).

Proposition 1 ([5]). *Let $\bar{\sigma}$ be a strategy profile in (G, v_0). Then $\bar{\sigma}$ is a weak SPE iff $\bar{\sigma}$ is a very weak SPE. There exists an initialized game (G, v_0) with a weak SPE but no SPE.*

[4] In this article, we will always use notation $\mu(h\rho)$ instead of $\mu_{|h}(\rho)$.

Fig. 1. An initialized game (G, v_0) with a (very) weak SPE and no SPE.

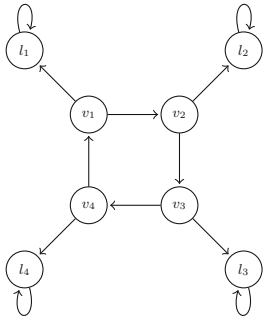


Fig. 2. Game G_4

Example 2 ([5]). Consider the two-player game (G, v_0) in Fig. 1 such that player 1 (resp. player 2) controls vertices v_0, v_2, v_3 (resp. vertex v_1). The set O of outcomes is equal to $\{o_1, o_2, o_3\}$, and the outcome function is prefix-independent such that $\mu((v_0v_1)^\omega) = o_1$, $\mu(v_2^\omega) = o_2$, and $\mu(v_3^\omega) = o_3$. The preference relation for player 1 (resp. player 2) is $o_1 \prec_1 o_2 \prec_1 o_3$ (resp. $o_2 \prec_2 o_3 \prec_2 o_1$).

It is known that this game has no SPE [26]. Nevertheless the strategy profile $\bar{\sigma}$ depicted with thick edges is a very weak SPE, and thus a weak SPE by Proposition 1. Let us give some explanation. Due to the simple form of the game, only two cases are to be treated. Consider first the subgame $(G_{|h}, v_0)$ with $h \in (v_0v_1)^*$, and the one-shot deviating strategy σ_1' from $\sigma_{1|h}$ such that $\sigma_1'(v_0) = v_2$. Then $\langle \bar{\sigma}_{|h} \rangle_{v_0} = v_0v_1v_3^\omega$ and $\langle \sigma_1', \sigma_{2|h} \rangle_{v_0} = v_0v_2^\omega$ with respective outcomes o_3 and o_2, showing that σ_1' is not a profitable deviation for player 1 in $(G_{|h}, v_0)$. Now in the subgame $(G_{|h}, v_1)$ with $h \in (v_0v_1)^*v_0$, the one-shot deviating strategy from $\sigma_{2|h}$ such that $\sigma_2'(v_1) = v_0$ is not profitable for player 2 in $(G_{|h}, v_1)$ because $\langle \bar{\sigma}_{|h} \rangle_{v_1} = v_1v_3^\omega$ and $\langle \sigma_{1|h}, \sigma_2' \rangle_{v_1} = v_1v_0v_1v_3^\omega$ with the same outcome o_3.

Notice that $\bar{\sigma}$ is not an SPE. Indeed the strategy σ_2' such that $\sigma_2'(hv_1) = v_0$ for all h, is infinitely deviating from σ_2, and is a profitable deviation for player 2 in (G, v_0) since $\langle \sigma_1, \sigma_2' \rangle_{v_0} = (v_0v_1)^\omega$ with outcome o_1.

3 General Conditions for the Existence of Weak SPEs

In this section, we propose general conditions to guarantee the existence of weak SPEs. In the next section, from this result, we will derive two interesting large families of games always having a weak SPE.

Theorem 1. *Let (G, v_0) be an initialized game with a subset $L \subseteq V$ of vertices called* leaves *with only one outgoing edge (l, l) for all $l \in L$. Suppose that:*

1. *for all $v \in V$, there exists a play $\rho = hl^\omega$ for some $h \in Hist(v)$ and $l \in L$,*
2. *for all plays $\rho = hl^\omega$ with $h \in Hist(v)$ and $l \in L$, $\mu(\rho) = \mu(l^\omega)$,*
3. *the set of outcomes $O_L = \{\mu(l^\omega) \mid l \in L\}$ is finite.*

Then there always exists a weak SPE $\bar{\sigma}$ in (G, v_0). Moreover, $\bar{\sigma}$ is finite-memory with memory size bounded by $|O_L|$.

Let us comment the hypotheses. The first condition means that from each vertex v of the game there is a leaf reachable from v; in particular L is not empty. The second condition expresses a prefix-independence of the outcome function restricted to plays eventually looping in a leaf $l \in L$. The last condition means that even if there is an infinite number of leaves, the set of outcomes assigned by μ to plays eventually looping in L is finite. The next example describes a family of games satisfying the conditions of Theorem 1.

Example 3. For each natural number $n \geq 3$, we build a game G_n with n players, $2n$ vertices, $3n$ edges, and $n+1$ outcomes. The set of players is $\Pi = \{1, 2, \ldots, n\}$ and the set of vertices is $V = \{v_1, \ldots, v_n, l_1, \ldots l_n\}$ such that $V_i = \{v_i, l_i\}$ for all $i \in \Pi$. The edges are $(v_1, v_2), (v_2, v_3), \ldots, (v_n, v_1)$, and $(v_i, l_i), (l_i, l_i)$ for all $i \in \Pi$. The game G_4 is depicted in Fig. 2. The set O of outcomes is equal to $\{o_1, \ldots, o_n, \bot\}$, and the outcome function is prefix-independent such that $\mu((v_1 v_2 \ldots v_n)^\omega) = \bot$ and $\mu(l_i^\omega) = o_i$ for all $i \in \Pi$. Each player i has a preference relation $\prec_i$ satisfying $\bot \prec_i o_{i-1} \prec_i o_i \prec_i o_j$ for all $j \in \Pi \backslash \{i-1, i\}$ (with the convention that $o_0 = o_n$).

Each game (G_n, v_1) satisfies the hypotheses of Theorem 1 with $L = \{l_1, \ldots, l_n\}$ and thus has a finite-memory weak SPE. Such a strategy profile $\bar{\sigma}$ is depicted in Fig. 3 for $n = 4$ (see the thick edges on the unravelling of G_4 from the initial vertex v_1) and can be easily generalized to every $n \geq 3$. One verifies that this profile is a very weak SPE, and thus a weak SPE by Proposition 1. For all $i \in \Pi$, the strategy σ_i of player i is finite-memory with a memory size equal to $n-1$. Intuitively, along $(v_1 \ldots v_n)^\omega$, player i repeatedly produces one move (v_i, l_i) followed by $n-2$ moves (v_i, v_{i+1}). Hence the memory states of the Moore machine for σ_i are counters from 1 to $n-1$.

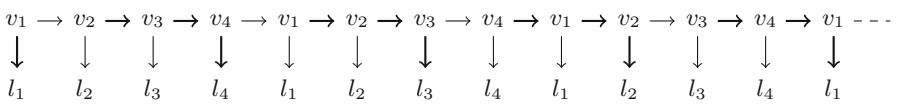


Fig. 3. Weak SPE in (G_4, v_1)

Let us now proceed to the proof of Theorem 1. Recall that it is enough to prove the existence of a very weak SPE by Proposition 1. The proof idea is the following one. Initially, for each vertex v, we accept all plays $\rho = hl^\omega$ with $h \in Hist(v)$ and $l \in L$ as *potential* plays induced by a very weak SPE in the initialized game (G, v). We thus label each v by the set of outcomes $\mu(l^\omega)$ for such leaves l (recall that $\mu(\rho) = \mu(l^\omega)$ by the second condition of Theorem 1). Notice that this labeling is finite (resp. not empty) by the third (resp. first) condition of the theorem. Step after step, we are going to remove some outcomes from the vertex labelings by a *Remove* operation followed by an *Adjust*operation.

The *Remove* operation removes an outcome o from the labeling of a given vertex v when there exists an edge (v, v') for which $o \prec_v o'$ for all outcomes o' that label v'. Indeed o cannot be the outcome of a play induced by a very weak SPE since the player who controls v will choose the move (v, v') to get a preferable outcome o'. Now it may happen that for another vertex u having o in its labeling, all potential plays induced by a very weak SPE from u with outcome o necessarily cross vertex v. As o has been removed from the labeling of v, these potential plays do no longer survive and o will also be removed from the labeling of u by the *Adjust* operation. Repeatedly applying these two operations converge to a fixpoint for which we will prove non-emptiness (this is the difficult part of the proof, non-emptiness will be obtained by maintaining three invariants, see Lemma 1). From this fixpoint, for each vertex v and each outcome o of the resulting labeling of v, there exists a play $\rho_{v,o} = hl^\omega$ with outcome o for some $h \in Hist(v)$ and $l \in L$. We can thus build a very weak SPE $\bar{\sigma}$ in (G, v_0) as follows. The construction of $\bar{\sigma}$ is done step by step: *(i)* initially $\bar{\sigma}$ is partially defined such that $\langle \bar{\sigma} \rangle_{v_0} = \rho_{v_0,o_0}$ for some o_0; *(ii)* then in the subgame $(G_{|h}, v)$ such that $\langle \bar{\sigma}_{|h} \rangle_v = \rho_{v,o}$, if the player who controls v chooses the move (v, v') in a one-shot deviation, then there exists $\rho_{v',o'}$ such that $o \not\prec_v o'$ by definition of the fixpoint, and we thus extend the construction of $\bar{\sigma}$ such that $\langle \bar{\sigma}_{|hv} \rangle_{v'} = \rho_{v',o'}$.

Let us now go into the details of the proof. For each $l \in L$, we denote by o_l the outcome $\mu(l^\omega)$. Recall that for all $\rho = hl^\omega$ we have $\mu(\rho) = o_l$ by the second hypothesis of the theorem. For each $v \in V$, we denote by $Succ(v)$ the set of successors of v distinct from v, that is, the vertices $v' \neq v$ such that $(v, v') \in E$. Notice that the leaves l are the vertices with only one outgoing edge (l, l). Thus, by definition, $Succ(v) = \emptyset$ for all $v \in L$ and $Succ(v) \neq \emptyset$ for all $v \in V \backslash L$.

The labeling $\lambda_\alpha(v)$ of the vertices v of G by subsets of O_L is an inductive process on the ordinal α. Initially (step $\alpha = 0$), each $v \in V$ is labeled by:

$$\lambda_0(v) = \{o_l \in O_L \mid \text{there exists a play } hl^\omega \text{ with } h \in Hist(v) \text{ and } l \in L\}.$$

(In particular $\lambda_0(l) = \{o_l\}$ for all $l \in L$). By the first hypothesis of the theorem, $\lambda_0(v) \neq \emptyset$. Let us introduce some additional terminology. At step α, when there is a path π from v to v' in G, we say that π is (o, α)-*labeled* if $o \in \lambda_\alpha(u)$ for all the vertices u of π. Thus initially, we have a $(o_l, 0)$-labeled path from v to l for each $o_l \in \lambda_0(v)$. For $v \in V$, let

$$m_\alpha(v) = \max_{\prec_v} \{\min_{\prec_v} \lambda_\alpha(v') \mid v' \in Succ(v)\}$$

with the convention that $m_\alpha(v) = \top$ if $Succ(v) = \emptyset$ or if $\lambda_\alpha(v') = \emptyset$ for all $v' \in Succ(v)$.[5] When $m_\alpha(v) \neq \top$, we says that $v' \in Succ(v)$ *realizes* $m_\alpha(v)$ if $m_\alpha(v) = \min_{\prec_v} \lambda_\alpha(v')$. Notice that even if $Succ(v)$ could be infinite, there are finitely many sets $\lambda_\alpha(v')$ since O_L is finite. This justifies our use of $\max_{\prec_v}$ and $\min_{\prec_v}$ operators in the definition of $m_\alpha(v)$.

[5] We suppose that $o \prec_v \top$ for all $o \in O_L$.

We alternate between *Remove* and *Adjust* that remove outcomes from labeling $\lambda_\alpha(v)$ in the following way:

– For an even[6] successor ordinal $\alpha + 2$,
  ***Remove* operation.** Test if for some $v \in V$, there exist $o \in \lambda_\alpha(v)$ and $v' \in Succ(v)$ such that

$$o \prec_v o', \text{for all} o' \in \lambda_\alpha(v').$$

  If such a v exists, then $\lambda_{\alpha+1}(v) = \lambda_\alpha(v) \backslash \{o\}$, and $\lambda_{\alpha+1}(u) = \lambda_\alpha(u)$ for the other vertices $u \neq v$. Otherwise $\lambda_{\alpha+1}(u) = \lambda_\alpha(u)$ for all $u \in V$.
  ***Adjust* operation.** Suppose that $\lambda_{\alpha+1}(v) = \lambda_\alpha(v) \backslash \{o\}$ at the previous step. For all $u \in V$ such that $o \in \lambda_{\alpha+1}(u)$, test if there exists a $(o, \alpha + 1)$-labeled path from u to some $l \in L$. If yes, then $\lambda_{\alpha+2}(u) = \lambda_{\alpha+1}(u)$, otherwise $\lambda_{\alpha+2}(u) = \lambda_{\alpha+1}(u) \backslash \{o\}$. For all $u \in V$ such that $o \notin \lambda_{\alpha+1}(u)$, let $\lambda_{\alpha+2}(u) = \lambda_{\alpha+1}(u)$.

  Suppose that $\lambda_{\alpha+1}(v) = \lambda_\alpha(v)$ for all $v \in V$ at the previous step, then $\lambda_{\alpha+2}(v) = \lambda_{\alpha+1}(v)$ for all $v \in V$.
  (Thus *Remove* is performed at odd step $\alpha + 1$, whereas *Adjust* is performed at even step $\alpha + 2$.)
– For a limit ordinal α, let $\lambda_\alpha(v) = \cap_{\beta<\alpha} \lambda_\beta(v)$ for all $v \in V$.

For each v, the sequence $(\lambda_\alpha(v))_\alpha$ is nonincreasing (for the set inclusion), and thus the sequence $(m_\alpha(v))_\alpha$ is nondecreasing (for the $\prec_v$ relation). Notice that for all leaves $l \in L$ and all steps α, we have $\lambda_\alpha(l) = \{o_l\}$. The next lemma states that we get a non empty fixpoint in the following sense:

Lemma 1. *There exists an ordinal α^* such that $\lambda_{\alpha^*}(v) = \lambda_{\alpha^*+1}(v) = \lambda_{\alpha^*+2}(v)$ for all $v \in V$. Moreover, $\lambda_{\alpha^*}(v) \neq \emptyset$ for all $v \in V$.*

Proof. Each set $\lambda_\alpha(v)$ has size bounded by $|O_L|$. During the inductive process, from step α (with α even) to step $\alpha+1$, *Remove* removes one outcome from one of these sets, and from step $\alpha + 1$ to step $\alpha + 2$, *Adjust* can remove outcomes from several such sets (it can remove no outcome at all). Therefore there exists an ordinal α^* such that $\lambda_{\alpha^*}(v) = \lambda_{\alpha^*+1}(v) = \lambda_{\alpha^*+2}(v)$ for all $v \in V$, and a fixpoint is then reached.[7] To prove that $\lambda_{\alpha^*}(v) \neq \emptyset$, we consider the next three invariants for which we will briefly explain that they are initially true and remain true after each step α. The non emptiness of $\lambda_{\alpha^*}(v)$ will follow from the second invariant.

INV1. For $v \in V$, we have for all $v' \in Succ(v)$ that

$$\{o \in \lambda_\alpha(v') \mid m_\alpha(v) \preceq_v o\} \subseteq \lambda_\alpha(v).$$

In particular, when $m_\alpha(v) \neq \top$, for each v' that realizes $m_\alpha(v)$, we have

$$\lambda_\alpha(v') \subseteq \lambda_\alpha(v). \tag{1}$$

[6] Ordinal 0 and each limit ordinal are even, and each successor ordinal $\alpha + 1$ is even (resp. odd) if α is odd (resp. even).

[7] When V is finite, it is reached after at most $2|O_L| \cdot |V|$ steps.

INV2. For $v \in V$, $\lambda_\alpha(v) \neq \emptyset$.
INV3. For $v \in V$, there exists a path from v to some $l \in L$ such that for all vertices u in this path, $\lambda_\alpha(u) \subseteq \lambda_\alpha(v)$.

The three invariants are initially true. Consider a limit-ordinal step α. Such a step is not explicitly removing outcomes, it is only summarizing what has been removed for lesser ordinals. Indeed for each vertex v, since the sets $\lambda_\beta(v)$ are finite, there is a last outcome removal occurring at some step $\beta < \alpha$. This helps proving that the invariants are indeed preserved at ordinal steps. The successor-ordinal steps are the difficult ones. The detailed proof invokes many times that the $\lambda_\alpha(v)$ and $m_\alpha(v)$ are monotone with respect to α.

Consider odd step $\alpha+1$ and the *Remove* operation. *(i) Remove* may remove from $\lambda_\alpha(v)$ only outcomes less than $m_\alpha(v)$, so it preserves INV1. *(ii) Remove* may remove only one outcome at only one vertex, so it preserves INV2 by (1). *(iii) Remove* preserves INV3. Indeed first note that *Remove* might only hurt INV3 at the vertex v subject to outcome removal. Let $v' \in Succ(v)$ that realizes $m_{\alpha+1}(v)$. By INV3 at step α there is a suitable path from v' to a leaf. Prefixing this path with v witnesses INV3 at step $\alpha+1$, using (1).

Consider even step $\alpha+2$ and the *Adjust* operation. *(i)* One checks that *Adjust* preserves INV1 by case splitting on whether $\lambda_{\alpha+2}(v) = \lambda_{\alpha+1}(v)$. *(ii)* By contradiction assume that $\lambda_{\alpha+1}(v) = \{o\}$ from which *Adjust* removes o. By INV3 there would be at prior step one path to a leaf labelled all along with o only. Such labels cannot be removed, leading to a contradiction. *(iii) Adjust* preserves INV3. Indeed from a vertex u_1 let $u_1 \dots u_n$ be a suitable path at step $\alpha+1$. If it is no longer suitable at step $\alpha+2$, some o was removed from some proper prefix $u_1 \dots u_{i-1}$, *i.e.* $o \in \lambda_{\alpha+2}(u_i)$ but $o \notin \lambda_{\alpha+2}(u_{i-1})$, so $o \notin \lambda_{\alpha+1}(u_{i-1})$ by definition of *Adjust*. INV3 provides a suitable path (void of o) from u_{i-1} at step $\alpha+1$. Concatenating it with $u_1 \dots u_{i-1}$ witnesses INV3 at step $\alpha+2$. □

By the previous lemma, we have a fixpoint such that that $\lambda_{\alpha^*}(v) \neq \emptyset$ for all $v \in V$. Moreover by *Adjust*, for all $o \in \lambda_{\alpha^*}(v)$, there is a (o, α^*)-labeled path π from v to some $l \in L$ with $o_l = o$. We denote by $\rho_{v,o}$ the play $\pi l^\omega \in Plays(v)$:

$$\rho_{v,o} = \pi l^\omega. \tag{2}$$

(*) Recall that $\mu(\rho_{v,o}) = o_l$, and that $o_l \in \lambda_{\alpha^*}(u)$ for all vertices u in $\rho_{v,o}$.

To get Theorem 1, it remains to explain how to build a weak SPE $\bar{\sigma}$ from this fixpoint that is finite-memory.

Proof (of Theorem 1). The construction of $\bar{\sigma}$ will be done step by step thanks to a progressive labeling of the histories by outcomes in O_L and by using the plays $\rho_{v,o}$. This labeling $\kappa : Hist(v_0) \to O_L$ will allow to recover from history hv the outcome o of the play $\langle \bar{\sigma}_{|h} \rangle_v$ induced by $\bar{\sigma}$ in the subgame $(G_{|h}, v)$.

We start with history v_0 and any $o_0 \in \lambda_{\alpha^*}(v_0)$. Consider ρ_{v_0,o_0} as in (2). The strategy profile $\bar{\sigma}$ is partially built such that $\langle \bar{\sigma} \rangle_{v_0} = \rho_{v_0,o_0}$. The non empty prefixes g of ρ_{v_0,o_0} are all labeled with $\kappa(g) = o_0$.

At the following steps, we consider a history $h'v'$ that is not yet labeled, but such that $h' = hv$ has already been labeled by $\kappa(hv) = o$. The labeling of hv by

o means that $\bar{\sigma}$ has already been built to produce the play $\langle \bar{\sigma}_{|h} \rangle_v$ with outcome o in the subgame $(G_{|h}, v)$, such that $\langle \bar{\sigma}_{|h} \rangle_v$ is suffix of $\rho_{u,o}$ from some u. By (*) we have $o \in \lambda_{\alpha^*}(v)$. By the fixpoint and in particular by *Remove* (with $o \in \lambda_{\alpha^*}(v)$ and $v' \in Succ(v)$), there exists $o' \in \lambda_{\alpha^*}(v')$ such that

$$o \not\prec_v o'. \tag{3}$$

With $\rho_{v',o'}$ as in (2), we then extend the construction of $\bar{\sigma}$ such that $\langle \bar{\sigma}_{|h'} \rangle_{v'} = \rho_{v',o'}$, and for each non empty prefix g of $\rho_{v',o'}$, we label $h'g$ by $\kappa(h'g) = o'$ (notice that the prefixes of h' have already been labeled by choice of h'). This process is iterated to complete the construction of $\bar{\sigma}$.

Let us show that $\bar{\sigma}$ is a very weak SPE in (G, v_0). Consider a history $h' = hv \in Hist(v_0)$ with $v \in V_i$, and a one-shot deviating strategy σ'_i from $\sigma_{i|h}$ in the subgame $(G_{|h}, v)$. Let v' be such that $\sigma'_i(v) = v'$. By definition of $\bar{\sigma}$, we have $\kappa(hv) = o$ and $\kappa(h'v') = o'$ such that (3) holds. Let $\rho = \langle \bar{\sigma}_{|h} \rangle_v$ and $\rho' = \langle \bar{\sigma}_{|h'} \rangle_{v'}$. Then $o = \mu(h\rho)$ and $o' = \mu(hv\rho')$ by (*). By (3), σ'_i is not a profitable deviation for player i. Hence $\bar{\sigma}$ is a very weak SPE and thus a weak SPE by Proposition 1.

It remains to prove that $\bar{\sigma}$ is finite-memory by correctly choosing the plays $\rho_{v,o}$ of (2). Fix $o \in O_L$ and consider the set U_o of vertices v such that $o \in \lambda_{\alpha^*}(v)$. We choose the plays $\rho_{v,o} = \pi l^\omega$ for all $v \in U_o$, such that the set of associated finite paths πl forms a tree. Hence having o in memory, one can produce positionally each $\rho_{v,o}$ with $v \in U_o$. Thus the memory-size of $\bar{\sigma}$ is equal to O_L. □

The next corollary is an easy consequence of Theorem 1.

Corollary 1. *Let (G, v_0) be an initialized game with a subset $L \subseteq V$ of leaves*[8] *such that the underlying graph is a tree rooted at v_0. If (G, v_0) satisfies the first and third conditions of Theorem 1, then there is a positional weak SPE in (G, v_0).*

In the next sections, we present two large families of games for which there always exists a weak SPE, as a consequence of Theorem 1 and its Corollary 1.

4 First Application

We begin with the first application of the results of the previous section (more particularly Corollary 1): when an initialized game has an outcome function with finite range, then it always has a weak SPE.

Theorem 2. *Let (G, v_0) be an initialized game such the outcome function has finite range. Then there exists a weak SPE in (G, v_0).*

Let us comment this theorem. *(i)* Kuhn's theorem [16] states that there exists an SPE in all initialized games played on *finite trees* (note that in this particular case, the existence of a weak SPE is equivalent to the existence of an SPE).

[8] The existence of leaves l with a unique outgoing edge (l, l) is abusive since the graph is a tree: it should be understood as a unique infinite play from l.

Theorem 2 can be seen as a generalization of Kuhn's theorem: if we keep the outcome set finite, all initialized games (regardless of the underlying graph and the player set) have a weak SPE.*(ii)* Theorem 2 guarantees the existence of a weak SPE for games with *Boolean* objectives as presented in Example 1, since their outcome function μ has finite range. It is proved in [27] that each initialized game with a finite number of players and Borel objectives has an SPE and thus a weak SPE. We thus here extend the existence of a weak SPE to an infinite number of players. *(iii)* The next theorem is proved in [12] for outcome functions $\mu = (\mu_i)_{i \in \Pi}$ as presented in Example 1 and has strong relationship with Theorem 2. Recall that $\mu_i : Plays \to \mathbb{R}$ is *lower-semicontinuous* if whenever a sequence of plays $(\rho_n)_{n \in \mathbb{N}}$ converges to play $\rho = \lim_{n \to \infty} \rho_n$, then $\liminf_{n \to \infty} \mu_i(\rho_n) \geq \mu_i(\rho)$.

Theorem 3 ([12]). *Let (G, v_0) be an initialized game with a finite set Π of players and an outcome function $\mu = (\mu_i)_{i \in \Pi}$ such that each $\mu_i : Plays \to \mathbb{R}$ has finite range and is lower-semicontinuous. Then there exists an SPE in (G, v_0).*

As every weak SPE is an SPE in the case of lower-semicontinuous payoff functions μ_i [5], we recover the previous result with our Theorem 2, however with a set of players of any cardinality and general outcome functions $\mu : Plays \to O$. Even if it is not explicitly mentioned in [12], a close look at the details of the proof shows that the authors first show the existence of a weak SPE (without the hypothesis of lower-semicontinuity) and then show that it is indeed an SPE (thanks to this hypothesis). The first part of their proof could be replaced by ours which is simpler (indeed we remove outcomes from the sets $\lambda_\alpha(v)$ (see the proof of Theorem 1) whereas plays are removed in the inductive process of [12]).

Intermediate results. The proofs of Theorem 2 in this section and Theorem 4 in the next section require intermediate results that we now describe. We begin with the next lemma where the set $\mu^{-1}(o)$, with $o \in O$, is called *dense in* (G, v_0) if for all $h \in Hist(v_0)$, there exists ρ such that $h\rho$ is a play with outcome o.

Lemma 2. *Let (G, v_0) be an initialized game. If for some $o \in O$, the set $\mu^{-1}(o)$ is dense in (G, v_0), then there exists a weak SPE with outcome o in (G, v_0).*

Lemma 2 leads to the next corollary. This corollary will provide a first step towards Theorem 4; it is already interesting on its own right.

Corollary 2. *Let G be a game such that the underlying graph is strongly connected and the outcome function μ is prefix-independent.*

- *For all outcomes o such that $o = \mu(\rho)$ with $\rho \in Plays(v_0)$, there exists a weak SPE with outcome o in (G, v_0).*
- *There exists a uniform strategy profile $\bar{\sigma}$ and an outcome o such that for all $v \in V$ taken as initial vertex, $\bar{\sigma}$ is a weak SPE in (G, v) with outcome o.*

We end with a last lemma which indicates how to combine different weak SPEs into one weak SPE. It will be used in the proofs of Theorems 2 and 4.

Lemma 3. *Consider an initialized game (G, v_0) and a set of vertices $L \subseteq V$ such that for all $hl \in Hist(v_0)$ with $l \in L$, the subgame $(G_{|h}, l)$ has a weak SPE with outcome o_{hl}. Consider the initialized game (G', v_0) obtained from (G, v_0):*

- *by replacing all edges $(l, v) \in E$ by one edge (l, l), for all $l \in L$,*
- *and with outcome function μ' such that for all $\rho' \in Plays_{G'}(v_0)$, $\mu'(\rho') = o_{hl}$ if $\rho' = hl^\omega$ with $l \in L$ and $\mu'(\rho') = \mu(\rho')$ otherwise.*

If (G', v_0) has a weak SPE, then (G, v_0) has also a weak SPE.

Proof of Theorem 2. We now proceed to the proof of Theorem 2. W.l.o.g. we can suppose that the underlying graph of G is a tree rooted at v_0 (by unraveling this graph from v_0). The proof idea is to apply Lemma 3 the conditions of which will be satisfied thanks to Lemma 2 (to get weak SPEs on some subgames) and Corollary 1 (to get a weak SPE on (G', v_0)).

Proof (of Theorem 2). Let us reason on the unraveling of G from v_0. By hypothesis, the outcome function μ has finite range. We denote by O the finite set of its outcomes. We are going to show how to get (*) a weak SPE in each subgame $(G_{|h}, v)$ of (G, v_0) (and thus in (G, v_0) itself) by induction on the size of O.

The basic case of (*) is trivial since for all subgames of (G, v_0), each strategy profile is a weak SPE when μ has range one. Suppose that O has size at least two, and that (*) holds for smaller sizes. We are going to build a set L as required by Lemma 3 to get a weak SPE in (G, v_0) and thus also in each of its subgames.

Let $o \in O$ and set $L' = \emptyset$. Consider the subgame $(G_{|h}, v)$ with $hv \in Hist_G(v_0)$. Then either the set $\mu_{|h}^{-1}(o)$ is dense in $(G_{|h}, v)$, or it is not. In the first case, there exists a weak SPE in $(G_{|h}, v)$ by Lemma 2. We add v to L'. In the second case, as $\mu_{|h}^{-1}(o)$ is not dense, there exists a history $h'v'$ in $Hist(v)$ such that $\mu_{|h}(h'\rho) \neq o$ for all $\rho \in Plays(v')$. Therefore, in the subgame $(G_{|hh'}, v')$, as the range of the outcome function $\mu_{|hh'}$ is smaller, there exists a weak SPE in $(G_{|hh'}, v')$ by induction hypothesis. As in the first case, we add v' to L'.

We repeat this process for all $hv \in Hist(v_0)$. We then get the set $L \subseteq L'$ as required by Lemma 3 by only keeping[9] the vertices $v \in L'$ such the associated history hv contains no vertex of L' except v. For each subgame $(G_{|h}, v)$ with $v \in L$, we thus have a weak SPE. The game (G', v_0) as defined in Lemma 3 has also a weak SPE by Corollary 1. It thus follows by Lemma 3 that there exists a weak SPE in (G, v_0), and thus also in each of its subgames. □

5 Second Application

In this section, we present a second large family of games with a weak SPE, as another application of the general results of Sect. 3 (more particularly Theorem 1). This family is constituted with all games with a finite underlying graph and a prefix-independent outcome function.

[9] L is the prefix-free subset of L'.

Theorem 4. *Let (G, v_0) be an initialized game with a finite underlying graph and a prefix-independent outcome function. Then there is a weak SPE in (G, v_0).*

Let us comment this theorem. *(i)* It guarantees the existence of a weak SPE for classical games with *quantitative* objectives as presented in Example 1, such that their outcome function is prefix-independent. This is the case of *limsup* and *mean-payoff* functions (and their limit inferior counterparts). Recall that Example 2 (see also Fig. 1) provides a game with no SPE, where the payoff functions μ_i can be seen as either *limsup* or *mean-payoff* (or their limit inferior counterparts). *(ii)* Later in this section, we will show that under the hypotheses of Theorem 4, there always exists a weak SPE that is *finite-memory* (Corollary 3), and we will study in which cases it can be *positional* or even *uniform* (Theorem 5). *(iii)* The families of games of Theorems 2 and 4 are incomparable: Boolean reachability games are in the first family but not in the second one, and mean-payoff games are in the second family but not in the first one.

The proof of Theorem 4 follows the same structure as for Theorem 2. The idea is to apply Lemma 3 where L is the union of the bottom strongly connected components of the graph of G. The weak SPEs required by Lemma 3 exist on the subgames $(G_{|h}, l)$, $l \in L$, by Corollary 2, and on the game (G', v_0) by Theorem 1.

Discussion on the memory. First we make the statement of Theorem 4 more precise by guaranteeing the existence of a weak SPE with finite-memory. The necessity of memory is illustrated by the family of games G_n of Example 3.

Corollary 3. *Let (G, v_0) be an initialized game with a finite underlying graph and a prefix-independent outcome function. Then there is a finite-memory weak SPE in (G, v_0) with $O(m)$ memory size where m is the number of bottom strongly connected components of the graph. A memory size linear in m is necessary.*

Second we identify conditions on the preference relations of the players, as expressed in the next lemma, that guarantee the existence of a *uniform* (instead of finite-memory) weak SPE (see Theorem 5).

Lemma 4 (Lemma 4 of [18]). *Let O be a set of outcomes. Let $\prec_i \subseteq O \times O$ be a preference relation for all $i \in \Pi$. The following assertions are equivalent.*

- *For all $i, i' \in \Pi$ and all $o, p, q \in O$, we have $\neg(o \prec_i p \prec_i q \wedge q \prec_{i'} o \prec_{i'} p)$.*
- *There exist a partition $\{O_k\}_{k \in K}$ of O and a linear order $<$ over K such that*
  - *$k < k'$ implies $o \prec_i o'$ for all $i \in \Pi$, $o \in O_k$ and $o' \in O_{k'}$,*
  - *$\prec_{i|O_k} = \prec_{i'|O_k}$ or $\prec_{i|O_k} = (\prec_{i'|O_k})^{-1}$ for all $i, i' \in \Pi$.*

In this lemma, we call each set O_k a *layer*. The second assertion states that (i) if $k < k'$ then all outcomes in $O_{k'}$ are preferred to all outcomes in O_k by all players, and (ii) inside a layer, any two players have either the same preference relations or the inverse ones. A set of outcomes satisfying the conditions of Lemma 4 is called *layered*. In [18], the author characterizes the preference relations that always yield SPE in games with outcome functions in the Hausdorff difference hierarchy of the open sets. One condition is that the set of outcomes is layered.

Theorem 5. *Let G be a game with a finite underlying graph and such that the outcome function is prefix-independent with a layered set O outcomes. Then there exists a uniform weak SPE in (G, v), for all $v \in V$.*

Example 4. Remember the class G_n of games, $n \geq 3$, of Example 3, such that $O = \{o_1, \ldots, o_n, \bot\}$ and each player i has a preference relation $\prec_i$ satisfying $\bot \prec_i o_{i-1} \prec_i o_i \prec_i o_j$ for all $j \in \Pi \backslash \{i-1, i\}$. This set of outcomes is not layered because the first assertion of Lemma 4 is not satisfied. Indeed we have $o_2 \prec_3 o_3 \prec_3 o_1$ and $o_1 \prec_2 o_2 \prec_2 o_3$. Recall that all weak SPEs of the games G_n require a memory size in $O(n)$ (by Corollary 3). Hence the hypothesis of Theorem 5 about the preference relations is not completely dispensable.

References

1. Berwanger, D.: Admissibility in infinite games. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393. Springer, Heidelberg (2007)
2. Brenguier, R., Clemente, L., Hunter, P., Pérez, G.A., Randour, M., Raskin, J.-F., Sankur, O., Sassolas, M.: Non-zero sum games for reactive synthesis. In: Dediu, A.-H., Janoušek, J., Martín-Vide, C., Truthe, B. (eds.) LATA 2016. LNCS, vol. 9618, pp. 3–23. Springer, Cham (2016). doi:10.1007/978-3-319-30000-9_1
3. Brenguier, R., Raskin, J.-F., Sankur, O.: Assume-admissible synthesis. In: CONCUR, LIPIcs, vol. 42, pp. 100–113. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2015)
4. Brenguier, R., Raskin, J.-F., Sassolas, M.: The complexity of admissibility in omega-regular games. In: CSL-LICS, pp. 23:1–23:10. ACM (2014)
5. Brihaye, T., Bruyère, V., Meunier, N., Raskin, J.-F.: Weak subgame perfect equilibria and their application to quantitative reachability. In: CSL, LIPIcs, vol. 41, pp. 504–518. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2015)
6. Bruyère, V., Meunier, N., Raskin, J.-F.: Secure equilibria in weighted games. In: CSL-LICS, pp. 26:1–26:26. ACM (2014)
7. Chatterjee, K., Henzinger, T.A., Jurdzinski, M.: Games with secure equilibria. Theor. Comput. Sci. **365**, 67–82 (2006)
8. Chatterjee, K., Doyen, L., Filiot, E., Raskin, J.-F.: Doomsday equilibria for omega-regular games. In: McMillan, K.L., Rival, X. (eds.) VMCAI 2014. LNCS, vol. 8318, pp. 78–97. Springer, Heidelberg (2014). doi:10.1007/978-3-642-54013-4_5
9. Chatterjee, K., Doyen, L., Henzinger, T.A.: Quantitative languages. ACM Trans. Comput. Log. **11**, 23:1–23:38 (2010)
10. Chatterjee, K., Henzinger, T.A.: Assume-guarantee synthesis. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 261–275. Springer, Heidelberg (2007). doi:10.1007/978-3-540-71209-1_21
11. Pril, J., Flesch, J., Kuipers, J., Schoenmakers, G., Vrieze, K.: Existence of secure equilibrium in multi-player games with perfect information. In: Csuhaj-Varjú, E., Dietzfelbinger, M., Ésik, Z. (eds.) MFCS 2014. LNCS, vol. 8635, pp. 213–225. Springer, Heidelberg (2014). doi:10.1007/978-3-662-44465-8_19
12. Flesch, J., Kuipers, J., Mashiah-Yaakovi, A., Schoenmakers, G., Solan, E., Vrieze, K.: Perfect-information games with lower-semicontinuous payoffs. Math. Oper. Res. **35**, 742–755 (2010)
13. Flesch, J., Predtetchinski, A.: A characterization of subgame perfect equilibrium plays in borel games of perfect information. Math. Oper. Res. (2017, to appear)

14. Fudenberg, D., Levine, D.: Subgame-perfect equilibria of finite- and infinite-horizon games. J. Econ. Theor. **31**, 251–268 (1983)
15. Grädel, E., Ummels, M.: Solution concepts and algorithms for infinite multiplayer games. In: New Perspectives on Games and Interaction, vol. 4, pp. 151–178. University Press, Amsterdam (2008)
16. Kuhn, H.W.: Extensive games and the problem of information, pp. 46–68. Classics in Game Theory (1953)
17. Kupferman, O., Perelli, G., Vardi, M.Y.: Synthesis with rational environments. Ann. Math. Artif. Intell. **78**(1), 3–20 (2016)
18. Le Roux, S.: Infinite subgame perfect equilibrium in the Hausdorff difference. In: Hajiaghayi, M.T., Mousavi, M.R. (eds.) TTCS 2015. LNCS, vol. 9541, pp. 147–163. Springer, Cham (2016)
19. Le Roux, S., Pauly, A.: Infinite sequential games with real-valued payoffs. In: CSL-LICS, pp. 62:1–62:10. ACM (2014)
20. Nash, J.F.: Equilibrium points in n-person games. In: PNAS, vol. 36, pp. 48–49. National Academy of Sciences (1950)
21. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: POPL, pp. 179–190. ACM Press (1989)
22. Purves, R.A., Sudderth, W.D.: Perfect information games with upper semicontinuous payoffs. Math. Oper. Res. **36**(3), 468–473 (2011)
23. Rubinstein, A.: Comments on the interpretation of game theory. Econometrica **59**, 909–924 (1991)
24. Selten, R.: Spieltheoretische Behandlung eines Oligopolmodells mit Nachfrageträgheit. Zeitschrift für die gesamte Staatswissenschaft **121**, 301–324, 667–689 (1965)
25. Shen, X.S., Yu, H., Buford, J., Akon, M.: Handbook of Peer-to-Peer Networking. Springer, Heidelberg (2010)
26. Solan, E., Vieille, N.: Deterministic multi-player Dynkin games. J. Math. Econ. **39**, 911–929 (2003)
27. 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). doi:10.1007/11944836_21

Optimal Reachability in Divergent Weighted Timed Games

Damien Busatto-Gaston(✉), Benjamin Monmege, and Pierre-Alain Reynier

Aix Marseille Univ, LIF, CNRS, Marseille, France
{damien.busatto,benjamin.monmege,pierre-alain.reynier}@lif.univ-mrs.fr

Abstract. Weighted timed games are played by two players on a timed automaton equipped with weights: one player wants to minimise the accumulated weight while reaching a target, while the other has an opposite objective. Used in a reactive synthesis perspective, this quantitative extension of timed games allows one to measure the quality of controllers. Weighted timed games are notoriously difficult and quickly undecidable, even when restricted to non-negative weights. Decidability results exist for subclasses of one-clock games, and for a subclass with non-negative weights defined by a semantical restriction on the weights of cycles. In this work, we introduce the class of *divergent weighted timed games* as a generalisation of this semantical restriction to arbitrary weights. We show how to compute their optimal value, yielding the first decidable class of weighted timed games with negative weights and an arbitrary number of clocks. In addition, we prove that divergence can be decided in polynomial space. Last, we prove that for untimed games, this restriction yields a class of games for which the value can be computed in polynomial time.

1 Introduction

Developing programs that verify real-time specifications is notoriously difficult, because such programs must take care of delicate timing issues, and are difficult to debug a posteriori. One research direction to ease the design of real-time software is to automatise the process. We model the situation into a timed game, played by a *controller* and an antagonistic *environment*: they act, in a turn-based fashion, over a *timed automaton* [2], namely a finite automaton equipped with real-valued variables, called clocks, evolving with a uniform rate. A usual objective for the controller is to reach a target. We are thus looking for a *strategy* of the controller, that is a recipe dictating how to play (timing delays and transitions to follow), so that the target is reached no matter how the environment plays. Reachability timed games are decidable [4], and EXPTIME-complete [21].

If the controller has a winning strategy in a given reachability timed game, several such winning strategies could exist. Weighted extensions of these games

The first author has been supported by ENS Cachan, Université Paris-Saclay. This work has been funded by the DeLTA project (ANR-16-CE40-0007), and by the SoSI project (PEPS SISC CNRS).

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 162–178, 2017.
DOI: 10.1007/978-3-662-54458-7_10

have been considered in order to measure the quality of the winning strategy for the controller [1,9]. This means that the game now takes place over a *weighted (or priced) timed automaton* [3,5], where transitions are equipped with weights, and states with rates of weights (the cost is then proportional to the time spent in this state, with the rate as proportional coefficient). While solving weighted timed automata has been shown to be PSPACE-complete [6] (i.e. the same complexity as the non-weighted version), weighted timed games are known to be undecidable [12]. This has led to many restrictions in order to regain decidability, the first and most interesting one being the class of strictly non-Zeno cost with only non-negative weights (in transitions and states) [1,9]: this hypothesis states that every execution of the timed automaton that follows a cycle of the region automaton has a weight far from 0 (in interval $[1, +\infty)$, for instance).

Less is known for weighted timed games in the presence of negative weights in transitions and/or states. In particular, no results exist so far for a class that does not restrict the number of clocks of the timed automaton to 1. However, negative weights are particularly interesting from a modelling perspective, for instance in case weights represent the consumption level of a resource (money, energy...) with the possibility to spend and gain some resource. In this work, we introduce a generalisation of the strictly non-Zeno cost hypothesis in the presence of negative weights, that we call *divergence*. We show the decidability of the class of divergent weighted timed games, with a 2-EXPTIME complexity (and an EXPTIME-hardness lower bound). These complexity results match the ones that could be obtained in the non-negative case from the study of [1,9].

Other types of payoffs than the accumulated weight we study (i.e. total payoff) have been considered for weighted timed games. For instance, energy and mean-payoff timed games have been introduced in [11]. They are also undecidable in general. Interestingly, a subclass called *robust timed games*, not far from our divergence hypothesis, admits decidability results. A weighted timed game is robust if, to say short, every simple cycle (cycle without repetition of a state) has weight non-negative or less than a constant $-\varepsilon$. Solving robust timed game can be done in EXPSPACE, and is EXPTIME-hard. Moreover, deciding if a weighted timed game is robust has complexity 2-EXPSPACE (and coNEXPTIME-hard). In contrast, we show that deciding the divergence of a weighted timed game is a PSPACE-complete problem.[1] In terms of modeling power, we do believe that divergence is sufficient for most cases. It has to be noted that extending our techniques and results in the case of robust timed games is intrinsically not possible: indeed, the value problem for this class is undecidable [10].

The property of divergence is also interesting in the absence of time. Indeed, weighted games with reachability objectives have been recently explored as a refinement of mean-payoff games [14,15]. A pseudo-polynomial time (i.e. polynomial if weights are encoded in unary) procedure has been proposed to solve them, and they are at least as hard as mean-payoff games. In this article, we also study divergent weighted games, and show that they are the first non-trivial class of

[1] Whereas all divergent weighted game are robust, the converse may not be true, since it is possible to mix positive and negative simple cycles in an SCC.

Table 1. Deciding weighted (timed) games with arbitrary weights

	Value of a game	Value of a divergent game	Deciding the divergence
Untimed	Pseudo-poly. [15]	PTIME-complete	NL (unary), PTIME (binary)
Timed	Undecidable [12]	2-EXPTIME, EXPTIME-hard	PSPACE-complete

weighted games with negative weights solvable in polynomial time. Table 1 summarises our results. We start in Sects. 2 and 3 by studying weighted (untimed) games, before considering the timed setting in Sects. 4 and 5. Complete proofs can be found in [17].

2 Weighted Games

We start our study with untimed games. We consider two-player turn-based games played on weighted graphs and denote the two *players* by Max and Min. A *weighted game*[2] is a tuple $\mathcal{G} = \langle V = V_{\mathsf{Min}} \uplus V_{\mathsf{Max}}, V_t, A, E, \mathsf{Weight}\rangle$ where V are vertices, partitioned into vertices belonging to Min (V_{Min}) and Max (V_{Max}), $V_t \subseteq V_{\mathsf{Min}}$ is a subset of target vertices for player Min, A is an alphabet, $E \subseteq V \times A \times V$ is a set of directed edges, and $\mathsf{Weight}\colon E \to \mathbb{Z}$ is the weight function, associating an integer weight with each edge. These games need not be finite in general, but in Sects. 2 and 3, we limit our study to the resolution of finite weighted games (where all previous sets are finite). We suppose that: (i) the game is deadlock-free, i.e. for each vertex $v \in V$, there is a letter $a \in A$ and a vertex $v' \in V$, such that $(v, a, v') \in E$; (ii) the game is deterministic, i.e. for each pair $(v, a) \in V \times A$, there is at most one vertex $v' \in V$ such that $(v, a, v') \in E$.[3]

A *finite play* is a finite sequence of edges $\rho = v_0 \xrightarrow{a_0} v_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{k-1}} v_k$, i.e. for all $0 \leqslant i < k$, $(v_i, a_i, v_{i+1}) \in E$. We denote by $|\rho|$ the length k of ρ. We often write $v_0 \xrightarrow{\rho} v_k$ to denote that ρ is a finite play from v_0 to v_k. The play ρ is said to be a *cycle* if $v_k = v_0$. We let $\mathsf{Plays}_{\mathcal{G}}$ be the set of all finite plays in $\mathcal{G}$, whereas $\mathsf{Plays}_{\mathcal{G}}^{\mathsf{Min}}$ and $\mathsf{Plays}_{\mathcal{G}}^{\mathsf{Max}}$ denote the finite plays that end in a vertex of Min and Max, respectively. A *play* is then an infinite sequence of consecutive edges.

A *strategy* for Min (respectively, Max) is a mapping $\sigma\colon \mathsf{Plays}_{\mathcal{G}}^{\mathsf{Min}} \to A$ (respectively, $\sigma\colon \mathsf{Plays}_{\mathcal{G}}^{\mathsf{Max}} \to A$) such that for all finite plays $\rho \in \mathsf{Plays}_{\mathcal{G}}^{\mathsf{Min}}$ (respectively, $\rho \in \mathsf{Plays}_{\mathcal{G}}^{\mathsf{Max}}$) ending in vertex v_k, there exists a vertex $v' \in V$ such that $(v_k, \sigma(\rho), v') \in E$. A play or finite play $\rho = v_0 \xrightarrow{a_0} v_1 \xrightarrow{a_1} \cdots$ conforms to a strategy σ of Min (respectively, Max) if for all k such that $v_k \in V_{\mathsf{Min}}$ (respectively, $v_k \in V_{\mathsf{Max}}$), we have that $a_k = \sigma(v_0 \xrightarrow{a_0} v_1 \cdots v_k)$. A strategy σ is *memoryless* if for all finite plays ρ, ρ' ending in the same vertex, we have that $\sigma(\rho) = \sigma(\rho')$. For all strategies σ_{Min} and σ_{Max} of players Min and Max, respectively, and for all vertices v, we let $\mathsf{Play}_{\mathcal{G}}(v, \sigma_{\mathsf{Max}}, \sigma_{\mathsf{Min}})$ be the outcome of σ_{Max} and σ_{Min}, defined as the unique play conforming to σ_{Max} and σ_{Min} and starting in v.

[2] Weighted games are called *min-cost reachability games* in [15].

[3] Actions are not standardly considered, but they become useful in the timed setting.

The objective of Min is to reach a target vertex, while minimising the accumulated weight up to the target. Hence, we associate to every finite play $\rho = v_0 \xrightarrow{a_0} v_1 \dots \xrightarrow{a_{k-1}} v_k$ its accumulated weight $\mathsf{Weight}_{\mathcal{G}}(\rho) = \sum_{i=0}^{k-1} \mathsf{Weight}(v_i, a_i, v_{i+1})$. Then, the weight of an infinite play $\rho = v_0 \xrightarrow{a_0} v_1 \xrightarrow{a_1} \cdots$, also denoted by $\mathsf{Weight}_{\mathcal{G}}(\rho)$, is defined by $+\infty$ if $v_k \notin V_t$ for all $k \geqslant 0$, or the weight of $v_0 \xrightarrow{a_0} v_1 \dots \xrightarrow{a_{k-1}} v_k$ if k is the first index such that $v_k \in V_t$. Then, we let $\mathsf{Val}_{\mathcal{G}}(v, \sigma_{\mathsf{Min}})$ and $\mathsf{Val}_{\mathcal{G}}(v, \sigma_{\mathsf{Max}})$ be the respective values of the strategies:

$$\mathsf{Val}_{\mathcal{G}}(v, \sigma_{\mathsf{Min}}) = \sup_{\sigma_{\mathsf{Max}}} \mathsf{Weight}_{\mathcal{G}}(\mathsf{Play}(v, \sigma_{\mathsf{Max}}, \sigma_{\mathsf{Min}}))$$
$$\mathsf{Val}_{\mathcal{G}}(v, \sigma_{\mathsf{Max}}) = \inf_{\sigma_{\mathsf{Min}}} \mathsf{Weight}_{\mathcal{G}}(\mathsf{Play}(v, \sigma_{\mathsf{Max}}, \sigma_{\mathsf{Min}})).$$

Finally, for all vertices v, we let $\underline{\mathsf{Val}}_{\mathcal{G}}(v) = \sup_{\sigma_{\mathsf{Max}}} \mathsf{Val}_{\mathcal{G}}(v, \sigma_{\mathsf{Max}})$ and $\overline{\mathsf{Val}}_{\mathcal{G}}(v) = \inf_{\sigma_{\mathsf{Min}}} \mathsf{Val}_{\mathcal{G}}(v, \sigma_{\mathsf{Min}})$ be the *lower* and *upper values* of v, respectively. We may easily show that $\underline{\mathsf{Val}}_{\mathcal{G}}(v) \leqslant \overline{\mathsf{Val}}_{\mathcal{G}}(v)$ for all v. We say that strategies $\sigma^{\star}_{\mathsf{Min}}$ of Min and $\sigma^{\star}_{\mathsf{Max}}$ of Max are optimal if, for all vertices v, $\mathsf{Val}_{\mathcal{G}}(v, \sigma^{\star}_{\mathsf{Max}}) = \underline{\mathsf{Val}}_{\mathcal{G}}(v)$ and $\mathsf{Val}_{\mathcal{G}}(v, \sigma^{\star}_{\mathsf{Min}}) = \overline{\mathsf{Val}}_{\mathcal{G}}(v)$, respectively. We say that a game $\mathcal{G}$ is *determined* if for all vertices v, its lower and upper values are equal. In that case, we write $\mathsf{Val}_{\mathcal{G}}(v) = \underline{\mathsf{Val}}_{\mathcal{G}}(v) = \overline{\mathsf{Val}}_{\mathcal{G}}(v)$, and refer to it as the *value* of v in $\mathcal{G}$. Finite weighted games are known to be determined [15]. If the game is clear from the context, we may drop the index $\mathcal{G}$ from all previous notations.

Problems. We want to compute the value of a *finite* weighted game, as well as optimal strategies for both players, if they exist. The corresponding decision problem, called the *value problem*, asks whether $\mathsf{Val}_{\mathcal{G}}(v) \leqslant \alpha$, given a finite weighted game $\mathcal{G}$, one of its vertices v, and a threshold $\alpha \in \mathbb{Z} \cup \{-\infty, +\infty\}$.

Related work. The value problem is a generalisation of the classical shortest path problem in a weighted graph to the case of two-player games. If weights of edges are all non-negative, a generalised Dijkstra algorithm enables to solve it in polynomial time [22]. In the presence of negative weights, a pseudo-polynomial-time (i.e. polynomial with respect to the game where weights are stored in unary) solution has been given in [15], based on a fixed point computation with value iteration techniques. Moreover, the value problem with threshold $-\infty$ is shown to be in $\mathsf{NP} \cap \mathsf{coNP}$, and as hard as solving mean-payoff games.

3 Solving Divergent Weighted Games

Our first contribution is to solve in polynomial time the value problem, for a subclass of finite weighted games that we call *divergent*. To the best of our knowledge, this is the first attempt to solve a non-trivial class of weighted games with arbitrary weights in polynomial time. Moreover, the same core technique is used for the decidability result in the timed setting that we will present in the next sections. Let us first define the class of divergent weighted games:

Definition 1. *A weighted game $\mathcal{G}$ is divergent when every cycle ρ of $\mathcal{G}$ satisfies* $\mathsf{Weight}(\rho) \neq 0$.

Divergence is a property of the underlying weighted graph, independent from the repartition of vertices between players. The term *divergent* reflects that cycling in the game ultimately makes the accumulated weight grow in absolute value. We will first formalise this intuition by analysing the strongly connected components (SCC) of the graph structure of a divergent game (the repartition of vertices into players does not matter for the SCC decomposition). Based on this analysis, we will obtain the following results:

Theorem 1. *The value problem over finite divergent weighted games is* PTIME*-complete. Moreover, deciding if a given finite weighted game is divergent is an* NL*-complete problem when weights are encoded in unary, and* PTIME *when they are encoded in binary.*

SCC analysis. A play ρ in $\mathcal{G}$ is said to be positive (respectively, negative) if $\mathsf{Weight}(\rho) > 0$ (respectively, $\mathsf{Weight}(\rho) < 0$). It follows that a cycle in a divergent weighted game is either positive or negative. A cycle is said to be simple if no vertices are visited twice (except for the common vertex at the beginning and the end of the cycle). We will rely on the following characterisation of divergent games in terms of SCCs.

Proposition 1. *A weighted game $\mathcal{G}$ is divergent if and only if, in each SCC of $\mathcal{G}$, all simple cycles are either all positive, or all negative.*

Proof. Let us first suppose that $\mathcal{G}$ is divergent. By contradiction, consider a negative simple cycle ρ (of weight $-p < 0$) and a positive simple cycle ρ' (of weight $p' > 0$) in the same SCC. Let v and v' be respectively the first vertices of ρ and ρ'. By strong connectivity, there exists a finite play η from v to v' and a finite play η' from v' to v. Let us consider the cycle ρ'' obtained as the concatenation of η and η'. If ρ'' has weight $q > 0$, the cycle obtained by concatenating q times ρ and p times ρ'' has weight 0, which contradicts the divergence of $\mathcal{G}$. The same reasoning on ρ'' and ρ' proves that ρ'' can not be negative. Thus, ρ'' is a cycle of weight 0, which again contradicts the hypothesis.

Reciprocally, consider a cycle of $\mathcal{G}$. It can be decomposed into simple cycles, all belonging to the same SCC. Therefore they are all positive or all negative. As the accumulated weight of the cycle is the sum of the weights of these simple cycles, $\mathcal{G}$ is divergent. □

Computing the values. Consider a divergent weighted game $\mathcal{G}$. Let us start by observing that vertices with value $+\infty$ are those from which Min can not reach the target vertices: thus, they can be computed with the classical attractor algorithm, and we can safely remove them, without changing other values or optimal strategies. In the rest, we therefore assume all values to be in $\mathbb{Z} \cup \{-\infty\}$.

Our computation of the values relies on a value iteration algorithm to find the greatest fixed point of operator $\mathcal{F} \colon (\mathbb{Z} \cup \{-\infty, +\infty\})^V \to (\mathbb{Z} \cup \{-\infty, +\infty\})^V$, defined for every vector $\boldsymbol{x}$ by $\mathcal{F}(\boldsymbol{x})_v = 0$ if $v \in V_t$, and otherwise

$$\mathcal{F}(\boldsymbol{x})_v = \begin{cases} \min\limits_{e=(v,a,v')\in E} \mathsf{Weight}(e) + \boldsymbol{x}_{v'} & \text{if } v \in V_{\mathsf{Min}} \\ \max\limits_{e=(v,a,v')\in E} \mathsf{Weight}(e) + \boldsymbol{x}_{v'} & \text{if } v \in V_{\mathsf{Max}}. \end{cases}$$

Indeed, this greatest fixed point is known to be the vector of values of the game (see, e.g., [15, Corollary 11]). In [15], it is shown that, by initialising the iterative evaluation of $\mathcal{F}$ with the vector $\boldsymbol{x}^0$ mapping all vertices to $+\infty$, the computation terminates after a number of iterations pseudo-polynomial in $\mathcal{G}$ (i.e. polynomial in the number of vertices and the greatest weight in $\mathcal{G}$). For $i > 0$, we let $\boldsymbol{x}^i = \mathcal{F}(\boldsymbol{x}^{i-1})$. Notice that the sequence $(\boldsymbol{x}^i)_{i\in\mathbb{N}}$ is non-increasing, since $\mathcal{F}$ is a monotonic operator. Value iteration algorithms usually benefit from decomposing a game into SCCs (in polynomial time), considering them in a bottom-up fashion: starting with target vertices that have value 0, SCCs are then considered in inverse topological order since the values of vertices in an SCC only depend on values of vertices of greater SCCs (in topological order), that have been previously computed.

Example 1. Consider the weighted game of Fig. 1, where Min vertices are drawn with circles, and Max vertices with squares. Vertex v_t is the only target. Near each vertex is placed its value. For a given vector $\boldsymbol{x}$, we have $\mathcal{F}(\boldsymbol{x})_{v_8} = \min(0 + \boldsymbol{x}_{v_t}, -1 + \boldsymbol{x}_{v_9})$ and $\mathcal{F}(\boldsymbol{x})_{v_2} = \max(-2 + \boldsymbol{x}_{v_1}, -1 + \boldsymbol{x}_{v_3}, -10 + \boldsymbol{x}_{v_5})$. By a computation of the attractor of $\{v_t\}$ for Min, we obtain directly that v_4 and v_7 have value $+\infty$. The inverse topological order on SCCs prescribes then to compute first the values for the SCC $\{v_8, v_9\}$, with target vertex v_t associated with value 0. Then, we continue with SCC $\{v_6\}$, also keeping a new target vertex v_8 with (already computed) value 0. For the trivial SCC $\{v_5\}$, a single application of $\mathcal{F}$ suffices to compute the value. Finally, for the SCC $\{v_1, v_2, v_3, v_4\}$, we keep a new target vertex v_5 with value 1.[4] Notice that this game is divergent, since, in each SCC, all simple cycles have the same sign.

For a divergent game $\mathcal{G}$, Proposition 1 allows us to know in polynomial time if a given SCC is positive or negative, i.e. if all cycles it contains are positive or negative, respectively: it suffices to consider an arbitrary cycle of it, and compute its weight. A trivial SCC (i.e. with a single vertex and no edges) will be arbitrarily considered positive. We now explain how to compute in polynomial time the value of all vertices in a positive or negative SCC.

First, in case of a positive SCC, we show that:

Proposition 2. *The value iteration algorithm applied on a positive SCC with n vertices stabilises after at most n steps.*

Proof (inspired by techniques used in [9]). Let $W = \max_{e\in E} |\mathsf{Weight}(e)|$ be the greatest weight in the game. There are no negative cycles in the SCC, thus there

[4] This means that, in the definition of $\mathcal{F}$, a vertex v of V_t is indeed mapped to its previously computed value, not necessarily 0.

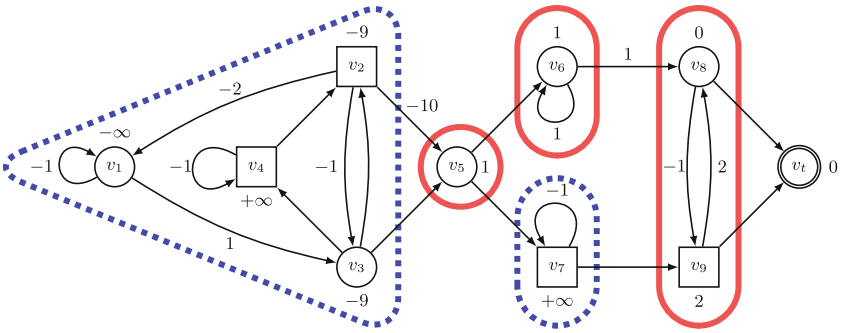


Fig. 1. SCC decomposition of a divergent weighted game: $\{v_1, v_2, v_3, v_4\}$ and $\{v_7\}$ are negative SCCs, $\{v_6\}$ and $\{v_8, v_9\}$ are positive SCCs, and $\{v_5\}$ is a trivial positive SCC.

are no vertices with value $-\infty$ in the SCC, and all values are finite. Let K be an upper bound on the values $|\boldsymbol{x}_v^n|$ obtained after n steps of the algorithm.[5] Fix an integer $p > (2K + W(n-1))n$. We will show that the values obtained after $n+p$ steps are identical to those obtained after n steps only. Therefore, since the algorithm computes non-increasing sequences of values, we have indeed stabilised after n steps only. Assume the existence of a vertex v such that $\boldsymbol{x}_v^{n+p} < \boldsymbol{x}_v^n$. By induction on p, we can show the existence of a vertex v' and a finite play ρ from v to v' with length p and weight $\boldsymbol{x}_v^{n+p} - \boldsymbol{x}_{v'}^n$: the play is composed of the edges that optimise successively the min/max operator in $\mathcal{F}$. This finite play being of length greater than $(2K+W(n-1))n$, there is at least one vertex appearing more than $2K + W(n-1)$ times. Thus, it can be decomposed into at least $2K + W(n-1)$ cycles and a finite play ρ' visiting each vertex at most once. All cycles of the SCC being positive, the weight of ρ is at least $2K + W(n-1) - (n-1)W = 2K$, bounding from below the weight of ρ' by $-(n-1)W$. Then, $\boldsymbol{x}_v^{n+p} - \boldsymbol{x}_{v'}^n \geqslant 2K$, so $\boldsymbol{x}_v^{n+p} \geqslant 2K + \boldsymbol{x}_{v'}^n \geqslant K$. But $K \geqslant \boldsymbol{x}_v^n$, so $\boldsymbol{x}_v^{n+p} \geqslant \boldsymbol{x}_v^n$, and that is a contradiction. □

Example 2. For the SCC $\{v_8, v_9\}$ of the game in Fig. 1, starting from $\boldsymbol{x}$ mapping v_8 and v_9 to $+\infty$, and v_t to 0, after one iteration, $\boldsymbol{x}_{v_8}$ changes for value 0, and after the second iteration, $\boldsymbol{x}_{v_9}$ stabilises to value 2.

Consider then the case of a negative SCC. Contrary to the previous case, we must deal with vertices of value $-\infty$. However, in a negative SCC, those vertices are easy to find[6]. These are all vertices where Max can not unilaterally guarantee to reach a target vertex:

[5] After n steps, the value iteration algorithm has set to a finite value all vertices, since it extends the attractor computation.

[6] This is in contrast with the general case of (non divergent) finite weighted games where the problem of deciding if a vertex has value $-\infty$ is as hard as solving mean-payoff games [15].

Proposition 3. *In a negative SCC with no vertices of value $+\infty$, vertices of value $-\infty$ are all the ones not in the attractor for* Max *to the targets.*

Proof. Consider a vertex v in the attractor for Max to the targets. Then, if Max applies a winning memoryless strategy for the reachability objective to the target vertices, all strategies of Min will generate a play from v reaching a target after at most $|V|$ steps. This implies that v has a finite (lower) value in the game.

Reciprocally, if v is not in the attractor, by determinacy of games with reachability objectives, Min has a (memoryless) strategy σ_{Min} to ensure that no strategy of Max permits to reach a target vertex from v. Applying σ_{Min} long enough to generate many negative cycles, before switching to a strategy allowing Min to reach the target (such a strategy exists since no vertex has value $+\infty$ in the game), allows Min to obtain from v a negative weight as small as possible. Thus, v has value $-\infty$. □

Thus, we can compute vertices of value $-\infty$ in polynomial time for a negative SCC. Then, finite values of other vertices can be computed in polynomial time with the following procedure. From a negative SCC $\mathcal{G}$ that has no more vertices of value $+\infty$ or $-\infty$, consider the dual (positive) SCC $\widetilde{\mathcal{G}}$ obtained by: (i) switching vertices of Min and Max; (ii) taking the opposite of every weight in edges. Sets of strategies of both players are exchanged in those two games, so that the upper value in $\mathcal{G}$ is equal to the opposite of the lower value in $\widetilde{\mathcal{G}}$, and vice versa. Since weighted games are determined, the value of $\mathcal{G}$ is the opposite of the value of $\widetilde{\mathcal{G}}$. Then, the value of $\mathcal{G}$ can be deduced from the value of $\widetilde{\mathcal{G}}$, for which Proposition 2 applies. We may also interpret this result as follows:

Proposition 4. *The value iteration algorithm, initialised with $\boldsymbol{x}_v^0 = -\infty$ (for all v), applied on a negative SCC with n vertices, and no vertices of value $+\infty$ or $-\infty$, stabilises after at most n steps.*

Proof. It is immediate that the vectors computed with this modified value iteration (that computes the smallest fixed point of $\mathcal{F}$) are exactly the opposite vectors of the ones computed in the dual positive SCC. The previous explanation is then a justification of the result. □

Example 3. Consider the SCC $\{v_1, v_2, v_3, v_4\}$ of the game in Fig. 1, where the value of vertex v_5 has been previously computed. We already know that v_4 has value $+\infty$ so we do not consider it further. The attractor of $\{v_5\}$ for Max is $\{v_2, v_3\}$, so that the value of v_1 is $-\infty$. Then, starting from $\boldsymbol{x}_0$ mapping v_2 and v_3 to $-\infty$, the value iteration algorithm computes this sequence of vectors: $\boldsymbol{x}_1 = (-9, -\infty)$ (Max tries to maximise the payoff, so he prefers to jump to the target to obtain $-10 + 1$ than going to v_3 where he gets $-1 - \infty$, while Min chooses v_2 to still guarantee $0 - \infty$), $\boldsymbol{x}_2 = (-9, -9)$ (now, Min has a choice between the target giving $0 + 1$ or v_3 giving $0 - 9$).

The proof for PTIME-hardness comes from a reduction (in logarithmic space) of the problem of solving finite games with reachability objectives [19]. To a reachability game, we simply add weights 1 on every transition, making it a

divergent weighted game. Then, Min wins the reachability game if and only if the value in the weighted game is lower than $|V|$.

In a divergent weighted game where all values are finite, optimal strategies exist. As observed in [15], Max always has a memoryless optimal strategy, whereas Min may require (finite) memory. Optimal strategies for both players can be obtained by combining optimal strategies in each SCC, the latter being obtained as explained in [15].

Class decision when weights are encoded in unary. We explain why deciding the divergence of a weighted game is an NL-complete problem. First, to prove the membership in NL, notice that a weighted game is *not divergent* if and only if there is a positive cycle and a negative cycle, both of length at most $|V|$, and belonging to the same SCC. To test this property in NL, we first guess a starting vertex for both cycles. Verifying that those are in the same SCC can be done in NL. Then, we guess the two cycles on-the-fly, keeping in memory their accumulated weights (smaller than $W \times |V|$, with W the biggest weight in the game, and thus of size at most logarithmic in the size of $\mathcal{G}$), and stop the on-the-fly exploration when the length of the cycles exceeds $|V|$. Therefore testing divergence is in $\mathsf{coNL} = \mathsf{NL}$ [20,25].

The NL-hardness (indeed coNL-hardness, which is equivalent [20,25]) is shown by a reduction of the reachability problem in a finite automaton. More precisely, we consider a finite automaton with a starting state and a different target state without outgoing transitions. We construct from it a weighted game by distributing all states to Min, and equipping all transitions with weight 1. We also add a loop with weight -1 on the target state and a transition from the target state to the initial state with weight 0. Then, the game is not divergent if and only if the target can be reached from the initial state in the automaton.

4 Weighted Timed Games

We now turn our attention to a timed extension of the weighted games. We will first define weighted timed games, giving their semantics in terms of *infinite* weighted games. We let X be a finite set of variables called clocks. A valuation of clocks is a mapping $\nu\colon X \to \mathbb{R}_{\geqslant 0}$. For a valuation ν, $d \in \mathbb{R}_{\geqslant 0}$ and $Y \subseteq X$, we define the valuation $\nu + d$ as $(\nu + d)(x) = \nu(x) + d$, for all $x \in X$, and the valuation $\nu[Y \leftarrow 0]$ as $(\nu[Y \leftarrow 0])(x) = 0$ if $x \in Y$, and $(\nu[Y \leftarrow 0])(x) = \nu(x)$ otherwise. The valuation $\mathbf{0}$ assigns 0 to every clock. A guard on clocks of X is a conjunction of atomic constraints of the form $x \bowtie c$, where $\bowtie \in \{\leqslant, <, =, >, \geqslant\}$ and $c \in \mathbb{N}$. A valuation $\nu\colon X \to \mathbb{R}_{\geqslant 0}$ satisfies an atomic constraint $x \bowtie c$ if $\nu(x) \bowtie c$. The satisfaction relation is extended to all guards g naturally, and denoted by $\nu \models g$. We let $G(X)$ the set of guards over X.

A weighted timed game is then a tuple $\mathcal{G} = \langle S = S_{\mathsf{Min}} \uplus S_{\mathsf{Max}}, S_t, \Delta, \mathsf{Weight}\rangle$ where S_{Min} and S_{Max} are *finite* disjoint subsets of states belonging to Min and Max, respectively, $S_t \subseteq S_{\mathsf{Min}}$ is a subset of target states for player Min, $\Delta \subseteq S \times G(X) \times 2^X \times S$ is a *finite* set of transitions, and $\mathsf{Weight}\colon \Delta \uplus S \to \mathbb{Z}$ is the weight function, associating an integer weight with each transition and state.

Without loss of generality, we may suppose that for each state $s \in S$ and valuation ν, there exists a transition $(s, g, Y, s') \in \Delta$ such that $\nu \models g$.

The semantics of a weighted timed game $\mathcal{G}$ is defined in terms of the infinite weighted game $\mathcal{H}$ whose vertices are configurations of the weighted timed game. A configuration is a pair (s, ν) with a state and a valuation of the clocks. Configurations are split into players according to the state. A configuration is final if its state is final. The alphabet of $\mathcal{H}$ is given by $\mathbb{R}_{\geqslant 0} \times \Delta$ and will encode the delay that a player wants to spend in the current state, before firing a certain transition. For every delay $d \in \mathbb{R}_{\geqslant 0}$, transition $\delta = (s, g, Y, s') \in \Delta$ and valuation ν, there is an edge $(s, \nu) \xrightarrow{d,\delta} (s', \nu')$ if $\nu + d \models g$ and $\nu' = (\nu + d)[Y \leftarrow 0]$. The weight of such an edge e is given by $d \times \mathsf{Weight}(s) + \mathsf{Weight}(\delta)$.

Plays, strategies, and values in the weighted timed game $\mathcal{G}$ are then defined as the ones in $\mathcal{H}$. It is known that weighted timed games are determined ($\underline{\mathsf{Val}}_{\mathcal{G}}(s, \nu) = \overline{\mathsf{Val}}_{\mathcal{G}}(s, \nu)$ for all state s and valuation ν).[7]

As usual in related work [1,9,10], we assume that all clocks are *bounded*, i.e. there is a constant $M \in \mathbb{N}$ such that every transition of the weighted timed games is equipped with a guard g such that $\nu \models g$ implies $\nu(x) \leqslant M$ for all clocks $x \in X$. We will rely on the crucial notion of regions, as introduced in the seminal work on timed automata [2]: a region is a set of valuations, that are all time-abstract bisimilar. There is only a finite number of regions and we denote by $\mathsf{Reg}(X, M)$ the set of regions associated with set of clocks X and maximal constant M in guards. For a valuation ν, we denote by $[\nu]$ the region that contains it. A region r' is said to be a time successor of region r if there exist $\nu \in r$, $\nu' \in r'$, and $d > 0$ such that $\nu' = \nu + d$. Moreover, for $Y \subseteq X$, we let $r[Y \leftarrow 0]$ be the region where clocks of Y are reset.

The region automaton $\mathcal{R}(\mathcal{G})$ of a game $\mathcal{G} = \langle S = S_{\mathsf{Min}} \uplus S_{\mathsf{Max}}, S_t, \Delta, \mathsf{Weight} \rangle$ is the finite automaton with states $S \times \mathsf{Reg}(X, M)$, alphabet Δ, and a transition $(s, r) \xrightarrow{\delta} (s', r')$ labelled by $\delta = (s, g, Y, s')$ if there exists a region r'' time successor of r such that r'' satisfies the guard g, and $r' = r''[Y \leftarrow 0]$. We call *path* an execution (not necessarily accepting) of this finite automaton, and we denote by π the paths. A play ρ in $\mathcal{G}$ is projected on a execution π in $\mathcal{R}(\mathcal{G})$, by replacing actual valuations by the regions containing them: we say that ρ *follows* path π. It is important to notice that, even if π is a cycle (i.e. starts and ends in the same state of the region automaton), there may exist plays following it in $\mathcal{G}$ that are not cycles, due to the fact that regions are sets of valuations.

Problems. As in weighted (untimed) games, we consider the *value problem*, mimicked from the one in $\mathcal{H}$. Precisely, given a weighted timed game $\mathcal{G}$, a configuration (s, ν) and a threshold $\alpha \in \mathbb{Z} \cup \{-\infty, +\infty\}$, we want to know whether $\mathsf{Val}_{\mathcal{G}}(s, \nu) \leqslant \alpha$. In the context of timed games, optimal strategies may not exist. We generally focus on ε-optimal strategies, that guarantee the optimal value, up to a small error ε.

[7] The result is stated in [13] for weighted timed games (called priced timed games) with one clock, but the proof does not use the assumption on the number of clocks.

Related work. In the one-player case, computing the optimal value and an ε-optimal strategy for weighted timed automata is known to be PSPACE-complete [6]. In the two-player case, much work for weighted timed games (also called priced timed games in the literature) has been achieved in the case of non-negative weights. In this setting, the value problem is undecidable [10,12]. To obtain decidability, one possibility is to limit the number of clocks to 1: then, there is an exponential-time algorithm to compute the value as well as ε-optimal strategies [7,18,24], whereas the problem is only known to be PTIME-hard. The other possibility to obtain a decidability result [1,9] is to enforce a semantical property of divergence, originally called strictly non-Zeno cost: it asks that every play following a cycle in the region automaton has weight at least 1.

In the presence of negative weights, undecidability even holds for weighted timed games with only 2 clocks [16] (for the existence problem asking if a strategy of player Min can guarantee a given threshold). Only the 1-clock restriction has been studied so far allowing one to obtain an exponential-time algorithm, under restrictions on the resets of the clock in cycles [13]. For weighted timed games, the strictly non-Zeno cost property has only been defined and studied in the absence of negative weights [9]. As already mentioned in the introduction, the notion is close, but not equivalent, to the one of robust weighted timed games, studied for mean-payoff and energy objectives [11]. In the next section, we extend the strictly non-Zeno cost property to negative weights calling it the divergence property, in order to obtain decidability of a large class of multi-clocks weighted timed games in the presence of arbitrary weights.

5 Solving Divergent Weighted Timed Games

We introduce divergent weighted timed games, as an extension of divergent weighted games to the timed setting.

Definition 2. *A weighted timed game $\mathcal{G}$ is divergent when every finite play ρ in $\mathcal{G}$ following a cycle in the region automaton $\mathcal{R}(\mathcal{G})$ satisfies* $\mathsf{Weight}(\rho) \notin (-1, 1)$.[8]

The weight is not only supposed to be different from 0, but also far from 0: otherwise, the original intuition on the ultimate growing of the values of plays would not be fulfilled. If $\mathcal{G}$ has only non-negative weights on states and transitions, this definition matches with the *strictly non-Zeno cost* property of [9, Theorem 6]. Our contributions summarise as follows:

Theorem 2. *The value problem over divergent weighted timed games is decidable in 2-EXPTIME, and is EXPTIME-hard. Moreover, deciding if a given weighted timed game is divergent is a PSPACE-complete problem.*

[8] As in [9], we could replace $(-1, 1)$ by $(-\kappa, \kappa)$ to define a notion of κ-divergence. However, since weights and guard constraints in weighted timed games are integers, for $\kappa \in (0, 1)$, a weighted timed game $\mathcal{G}$ is κ-divergent if and only if it is divergent.

Remember that these complexity results match the ones that can be obtained from the study of [9] for non-negative weights.

SCC analysis. Keeping the terminology of the untimed setting, a cycle π of $\mathcal{R}(\mathcal{G})$ is said to be positive (respectively, negative) if every play ρ following π satisfies $\mathsf{Weight}(\rho) \geqslant 1$ (respectively, $\mathsf{Weight}(\rho) \leqslant -1$). By definition, every cycle of the region automaton of a divergent weighted timed game is positive or negative. Moreover, notice that checking if a cycle π is positive or negative can be done in polynomial time with respect to the length of π. Indeed, the set $\{\mathsf{Weight}(\rho) \mid \rho \text{ is a play following } \pi\}$ is an interval, as the image of a convex set by an affine function (see [6, Sect. 3.2] for explanation), and the extremal points of this interval can be computed in polynomial time by solving a linear problem [6, Corollary 1]. We first transfer in the timed setting the characterisation of divergent games in terms of SCCs that we relied on in the untimed setting:

Proposition 5. *A weighted timed game $\mathcal{G}$ is divergent if and only if, in each SCC of $\mathcal{R}(\mathcal{G})$, simple cycles are either all positive, or all negative.*

The proof of the reciprocal follows the exact same reasoning as for weighted games (see Proposition 1). For the direct implication, the situation is more complex: we need to be more careful while composing cycles with each others, and weights in the timed game are no longer integers, forbidding the arithmetical reasoning we applied. To help us, we rely on the corner-point abstraction introduced in [8] to study multi-weighted timed automata. It consists in adding a weighted information to the edges $(s, r) \xrightarrow{\delta} (s', r')$ of the region automaton. Since the weights depend on the exact valuations ν and ν', taken in regions r and r', respectively, the weight of such an edge in the region automaton is computed for each pair of *corners* of the regions. Formally, corners of region r are valuations in $\overline{r} \cap \mathbb{N}^X$ (where $\overline{r}$ denotes the topological closure of r). Since corners do not necessarily belong to their regions, we must consider a modified version $\overline{\mathcal{G}}$ of the game $\mathcal{G}$ where all strict inequalities of guards have been replaced with non-strict ones. Then, for a path π in $\mathcal{R}(\mathcal{G})$, we denote by $\overline{\pi}$ the equivalent of path π in $\mathcal{R}(\overline{\mathcal{G}})$. In the following, our focus is on cycles of the region automaton, so we only need to consider the aggregation of all the behaviours following a cycle. Inspired by the *folded orbit graphs* (FOG) introduced in [23], we define the folded orbit graph $\mathsf{FOG}(\pi)$ of a cycle $\pi = (s_1, r = r_1) \xrightarrow{\delta_1} (s_2, r_2) \xrightarrow{\delta_2} \cdots \xrightarrow{\delta_n} (s_1, r)$ in $\mathcal{R}(\mathcal{G})$ as a graph whose vertices are corners of region r, and that contains an edge from corner v to corner v' if there exists a finite play $\overline{\rho}$ in $\overline{\mathcal{G}}$ from (s_1, v) to (s_1, v') following $\overline{\pi}$ jumping from corners to corners[9]. We fix such a finite play $\overline{\rho}$ arbitrarily and label the edge between v and v' in the FOG by this play: it is then denoted by $v \xrightarrow{\overline{\rho}} v'$. Moreover, since $\overline{\rho}$ jumps from corners to corners, its weight $\mathsf{Weight}(\overline{\rho})$ is an integer, conforming to the definitions of the corner-point abstraction of [8]. Following [8, Proposition 5], it is possible to find a play ρ in

[9] Notice that if there is a play from (s_1, v) to (s_1, v') in $\overline{\mathcal{G}}$, there is another one that only jumps at corners of regions.

$\mathcal{G}$ close to $\overline{\rho}$, in the sense that we control the difference between their respective weights:

Lemma 1. *For all $\varepsilon > 0$ and edge $v \xrightarrow{\overline{\rho}} v'$ of $\mathsf{FOG}(\pi)$, there exists a play ρ in $\mathcal{G}$ following π such that $|\mathsf{Weight}(\rho) - \mathsf{Weight}(\overline{\rho})| \leqslant \varepsilon$.*

In order to prove the direct implication of Proposition 5, suppose now that $\mathcal{G}$ is divergent, and consider two simple cycles π and π' in the same SCC of $\mathcal{R}(\mathcal{G})$. We need to show that they have the same sign. Lemma 2 will first take care of the case where π and π' share a state (s, r).

Lemma 2. *If $\mathcal{G}$ is divergent and two cycles π and π' of $\mathcal{R}(\mathcal{G})$ share a state (s, r), they are either both positive or both negative.*

Proof. Suppose by contradiction that π is negative and π' is positive. We assume that (s, r) is the first state of both π and π', possibly performing cyclic permutations of states if necessary. We construct a graph $\mathsf{FOG}(\pi, \pi')$ as the union of $\mathsf{FOG}(\pi)$ and $\mathsf{FOG}(\pi')$ (that share the same set of vertices), colouring in blue the edges of $\mathsf{FOG}(\pi)$ and in red the edges of $\mathsf{FOG}(\pi')$. A path in $\mathsf{FOG}(\pi, \pi')$ is said blue (respectively, red) when all of its edges are blue (respectively, red).

We assume first that there exists in $\mathsf{FOG}(\pi, \pi')$ a blue cycle C and a red cycle C' with the same first vertex v. Let k and k' be the respective lengths of C and C', so that C can be decomposed as $v \xrightarrow{\overline{\rho_1}} \cdots \xrightarrow{\overline{\rho_k}} v$ and C' as $v \xrightarrow{\overline{\rho'_1}} \cdots \xrightarrow{\overline{\rho'_{k'}}} v$, where $\overline{\rho_i}$ are plays following $\overline{\pi}$ and $\overline{\rho'_i}$ are plays following $\overline{\pi'}$, all jumping only on corners of regions. Let $\overline{\rho}$ be the concatenation of $\overline{\rho_1}, \ldots, \overline{\rho_k}$, and $\overline{\rho'}$ be the concatenation of $\overline{\rho'_1}, \ldots, \overline{\rho'_{k'}}$. Recall that $w = |\mathsf{Weight}(\overline{\rho})|$ and $w' = |\mathsf{Weight}(\overline{\rho'})|$ are integers. Since π is negative, so is π^k, the concatenation of k copies of π (the weight of a play following it is a sum of weights all below -1). Therefore, $\overline{\rho}$, that follows π^k, has a weight $\mathsf{Weight}(\overline{\rho}) \leqslant -1$. Similarly, $\mathsf{Weight}(\overline{\rho'}) \geqslant 1$. We consider the cycle C'' obtained by concatenating w' copies of C and w copies of C'. Similarly, we let $\overline{\rho''}$ be the play obtained by concatenating w' copies of $\overline{\rho}$ and w copies of $\overline{\rho'}$. By Lemma 1, there exists a play ρ'' in $\mathcal{G}$, following C'' such that $|\mathsf{Weight}(\rho'') - \mathsf{Weight}(\overline{\rho''})| \leqslant 1/3$. But $\mathsf{Weight}(\overline{\rho''}) = \mathsf{Weight}(\overline{\rho})w' + \mathsf{Weight}(\overline{\rho'})w = 0$, so $\mathsf{Weight}(\rho'') \in (-1, 1)$: this contradicts the divergence of $\mathcal{G}$, since ρ'' follows the cycle of $\mathcal{R}(\mathcal{G})$ composed of w' copies π^k and w copies of $\pi'^{k'}$ of $\mathcal{R}(\mathcal{G})$.

We now return to the general case, where C and C' may not exist. Since $\mathsf{FOG}(\pi)$ and $\mathsf{FOG}(\pi')$ are finite graphs with no deadlocks (every corner has an outgoing edge), from every corner of $\mathsf{FOG}(\pi, \pi')$, we can reach a blue simple cycle, as well as a red simple cycle. Since there are only a finite number of simple cycles in $\mathsf{FOG}(\pi, \pi')$, there exists a blue cycle C and a red cycle C' that can reach each other in $\mathsf{FOG}(\pi, \pi')$. In $\mathsf{FOG}(\pi, \pi')$, we let P be a path from the first vertex of C to the first vertex of C', and P' be a path from the first vertex of C' to the first vertex of C. Consider the cycle C'' obtained by concatenating P and P'. As a cycle of $\mathsf{FOG}(\pi, \pi')$, we can map it to a cycle π'' of $\mathcal{R}(\mathcal{G})$ (alternating π and π' depending on the colours of the traversed edges), so that C'' is a cycle (of length 1) of $\mathsf{FOG}(\pi'')$. By the divergence of $\mathcal{G}$, π'' is positive or negative.

Suppose for instance that it is positive. Since (s, r) is the first state of both π and π'', we can construct the $\mathsf{FOG}(\pi, \pi'')$, in which C is a blue cycle and C'' is a red cycle, both sharing the same first vertex. We then conclude with the previous case. A similar reasoning with π' applies to the case that π'' is negative. Therefore, in all cases, we reached a contradiction. □

To finish the proof of the direct implication of Proposition 5, we suppose that the two simple cycles π and π' in the same SCC of $\mathcal{R}(\mathcal{G})$ do not share any states. By strong connectivity, in $\mathcal{R}(\mathcal{G})$, there exists a path π_1 from the first state of π to the first state of π', and a path π_2 from the first state of π' to the first state of π. Consider the cycle of $\mathcal{R}(\mathcal{G})$ obtained by concatenating π_1 and π_2. By divergence of $\mathcal{G}$, it must be positive or negative. Since it shares a state with both π and π', Lemma 2 allows us to prove a contradiction in both cases. This concludes the proof of Proposition 5.

Value computation. We will now explain how to compute the values of a divergent weighted timed game $\mathcal{G}$. Remember that the function Val maps configurations of $S \times \mathbb{R}^X_{\geqslant 0}$ to a value in $\mathbb{R}_\infty = \mathbb{R} \cup \{-\infty, +\infty\}$. The semi-algorithm of [9] relies on the same principle as the value iteration algorithm used in the untimed setting, only this time we compute the greatest fixed point of operator $\mathcal{F} \colon \mathbb{R}_\infty^{S \times \mathbb{R}^X_{\geqslant 0}} \to \mathbb{R}_\infty^{S \times \mathbb{R}^X_{\geqslant 0}}$, defined by $\mathcal{F}(\boldsymbol{x})_{(s,\nu)} = 0$ if $s \in S_t$, and otherwise

$$\mathcal{F}(\boldsymbol{x})_{(s,\nu)} = \begin{cases} \sup\limits_{(s,\nu) \xrightarrow{d,\delta} (s',\nu')} d \times \mathsf{Weight}(s) + \mathsf{Weight}(\delta) + \boldsymbol{x}_{(s',\nu')} & \text{if } s \in S_{\mathsf{Max}} \\ \inf\limits_{(s,\nu) \xrightarrow{d,\delta} (s',\nu')} d \times \mathsf{Weight}(s) + \mathsf{Weight}(\delta) + \boldsymbol{x}_{(s',\nu')} & \text{if } s \in S_{\mathsf{Min}} \end{cases}$$

where $(s, \nu) \xrightarrow{d,\delta} (s', \nu')$ ranges over the edges of the infinite weighted game associated with $\mathcal{G}$ (the one defining its semantics). Then, starting from $\boldsymbol{x}^0$ mapping every configuration to $+\infty$, we let $\boldsymbol{x}^i = \mathcal{F}(\boldsymbol{x}^{i-1})$ for all $i > 0$. Since $\boldsymbol{x}^0$ is piecewise affine (even constant), and $\mathcal{F}$ preserves piecewise affinity, all iterates $\boldsymbol{x}^i$ are piecewise affine with a finite amount of pieces. In [1], it is proved that $\boldsymbol{x}^i$ has at most a number of pieces linear in the size of $\mathcal{R}(\mathcal{G})$ and exponential in i.[10]

First, we can compute the set of configurations having value $+\infty$. Indeed, the region automaton $\mathcal{R}(\mathcal{G})$ can be seen as a reachability two-player game $\mathcal{S}(\mathcal{G})$ by saying that (s, r) belongs to Min (Max, respectively) if $s \in S_{\mathsf{Min}}$ ($s \in S_{\mathsf{Max}}$, respectively). Notice that if $\mathsf{Val}(s, \nu) = +\infty$, then for all $\nu' \in [\nu], \mathsf{Val}(s, \nu') = +\infty$. Therefore, a configuration (s, ν) cannot reach the target states if and only if $(s, [\nu])$ is not in the attractor of Min to the targets in $\mathcal{S}(\mathcal{G})$. As a consequence, we can compute all such states of $\mathcal{S}(\mathcal{G})$ with complexity linear in the size of $\mathcal{R}(\mathcal{G})$.

We then decompose $\mathcal{R}(\mathcal{G})$ in SCCs. By Proposition 5, each SCC is either positive or negative (i.e. it contains only positive cycles, or only negative ones). Then, in order to find the sign of a component, it suffices to find one of its simple

[10] For divergent games with only non-negative weights, the fixed point is reached after a number of steps linear in the size of the region automaton [9]: overall, this leads to a doubly exponential complexity.

cycles, for example with a depth-first search, then compute the weight of one play following it.

As we did for weighted (untimed) games, we then compute values in inverse topological order over the SCCs. Once the values of all configurations in (s, r) appearing in previously considered SCCs have been computed, they are no longer modified in further computation. This is the case, in particular, for all pairs (s, r) that have value $+\infty$, that we precompute from the beginning. In order to resolve a positive SCC of $\mathcal{R}(\mathcal{G})$, we apply $\mathcal{F}$ on the current piecewise affine function, only modifying the pieces appearing in the SCC, until reaching a fixed point over these pieces. In order to resolve a negative SCC of $\mathcal{R}(\mathcal{G})$, we compute the attractor for Max to the previously computed SCCs: outside of this attractor, we set the value to $-\infty$. Then, we apply $\mathcal{F}$ for pieces appearing in the SCC, initialising them to $-\infty$ (equivalently, we compute in the dual game, that is a positive SCC), until reaching a fixed point over these pieces. The next proposition contains the correction and termination arguments that where presented in Propositions 2, 3, and 4 for the untimed setting:

Proposition 6. *Let $\mathcal{G}$ be a divergent game with no configurations of value $+\infty$.*

1. *The value iteration algorithm applied on a positive SCC of $\mathcal{R}(\mathcal{G})$ with n states stabilises after at most n steps.*
2. *In a negative SCC, states (s, r) of $\mathcal{R}(\mathcal{G})$ of value $-\infty$ are all the ones not in the attractor for Max to the targets.*
3. *The value iteration algorithm, initialised with $-\infty$, applied on a negative SCC of $\mathcal{R}(\mathcal{G})$ with n states, and no vertices of value $-\infty$, stabilises after at most n steps.*

By the complexity results of [1, Theorem 3], we obtain a doubly exponential time algorithm computing the value of a divergent weighted timed game. This shows that the value problem is in 2-$\mathsf{EXPTIME}$ for divergent weighted timed game. The proof for $\mathsf{EXPTIME}$-hardness comes from a reduction of the problem of solving timed games with reachability objectives [21]. To a reachability timed game, we simply add weights 1 on every transition and 0 on every state, making it a divergent weighted timed game. Then, Min wins the reachability timed game if and only if the value in the weighted timed game is lower than threshold $\alpha = |S| \times |\mathsf{Reg}(X, M)|$.

In an SCC of $\mathcal{R}(\mathcal{G})$, the value iteration algorithm of [1] allows us to compute an ε-optimal strategy for both players (for configurations having a finite value), that is constant (delay or fire a transition) over each piece of the piecewise affine value function. As in the untimed setting, we may then compose such ε-optimal strategies to obtain an ε'-optimal strategy in $\mathcal{G}$ (ε' is greater than ε, but can be controlled with respect to the number of SCCs in $\mathcal{R}(\mathcal{G})$).

Class decision. Deciding if a weighted timed game is divergent is PSPACE-complete. The proof is an extension of the untimed setting NL-complete result, but this time we reason on regions, hence the exponential blowup in complexity: it heavily relies on Proposition 5, as well as the corner-point abstraction to keep a compact representation of plays.

6 Conclusion

In this article, we introduced the first decidable class of weighted timed games with arbitrary weights, with no restrictions on the number of clocks. Future work include the approximation problem for a larger class of weighted timed games (divergent ones where we also allow cycles of weight exactly 0), already studied with only non-negative weights by [10].

References

1. Alur, R., Bernadsky, M., Madhusudan, P.: Optimal reachability for weighted timed games. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 122–133. Springer, Heidelberg (2004). doi:10.1007/978-3-540-27836-8_13
2. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994)
3. Alur, R., Torre, S., Pappas, G.J.: Optimal paths in weighted timed automata. Theoret. Comput. Sci. **318**(3), 297–322 (2004)
4. Asarin, E., Maler, O.: As soon as possible: time optimal control for timed automata. In: Vaandrager, F.W., Schuppen, J.H. (eds.) HSCC 1999. LNCS, vol. 1569, pp. 19–30. Springer, Heidelberg (1999). doi:10.1007/3-540-48983-5_6
5. Behrmann, G., Fehnker, A., Hune, T., Larsen, K., Pettersson, P., Romijn, J., Vaandrager, F.: Minimum-cost reachability for priced time automata. In: Benedetto, M.D., Sangiovanni-Vincentelli, A. (eds.) HSCC 2001. LNCS, vol. 2034, pp. 147–161. Springer, Heidelberg (2001). doi:10.1007/3-540-45351-2_15
6. Bouyer, P., Brihaye, T., Bruyère, V., Raskin, J.-F.: On the optimal reachability problem of weighted timed automata. Formal Meth. Syst. Des. **31**(2), 135–175 (2007)
7. Bouyer, P., Brihaye, T., Markey, N.: Improved undecidability results on weighted timed automata. Inf. Process. Lett. **98**(5), 188–194 (2006)
8. Bouyer, P., Brinksma, E.D., Larsen, K.G.: Optimal infinite scheduling for multi-priced timed automata. Formal Methods in System Design **32**(1), 3–23 (2008)
9. Bouyer, P., Cassez, F., Fleury, E., Larsen, K.G.: Optimal strategies in priced timed game automata. In: Lodaya, K., Mahajan, M. (eds.) FSTTCS 2004. LNCS, vol. 3328, pp. 148–160. Springer, Heidelberg (2004). doi:10.1007/978-3-540-30538-5_13
10. Bouyer, P., Jaziri, S., Markey, N.: On the value problem in weighted timed games. In: Proceedings of the 26th International Conference on Concurrency Theory (CONCUR 2015), vol. 42 of Leibniz International Proceedings in Informatics, pp. 311–324. Leibniz-Zentrum für Informatik (2015)
11. Brenguier, R., Cassez, F., Raskin, J.-F.: Energy, mean-payoff timed games. In: Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control (HSCC 2014), pp. 283–292. ACM (2014)
12. Brihaye, T., Bruyère, V., Raskin, J.-F.: On optimal timed strategies. In: Pettersson, P., Yi, W. (eds.) FORMATS 2005. LNCS, vol. 3829, pp. 49–64. Springer, Heidelberg (2005). doi:10.1007/11603009_5
13. Brihaye, T., Geeraerts, G., Haddad, A., Lefaucheux, E., Monmege, B.: Simple priced timed games are not that simple. In: Proceedings of the 35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015), vol. 45 of LIPIcs, pp. 278–292. Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2015)

14. Brihaye, T., Geeraerts, G., Haddad, A., Monmege, B.: To reach or not to reach? Efficient algorithms for total-payoff games. In: Proceedings of the 26th International Conference on Concurrency Theory (CONCUR'15), vol. 42 of LIPIcs, pp. 297–310. Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2015)
15. Brihaye, T., Geeraerts, G., Haddad, A., Monmege, B.: Pseudopolynomial iterative algorithm to solve total-payoff games and min-cost reachability games. Acta Informatica (2016)
16. Brihaye, T., Geeraerts, G., Narayanan Krishna, S., Manasa, L., Monmege, B., Trivedi, A.: Adding negative prices to priced timed games. In: Baldan, P., Gorla, D. (eds.) CONCUR 2014. LNCS, vol. 8704, pp. 560–575. Springer, Heidelberg (2014). doi:10.1007/978-3-662-44584-6_38
17. Busatto-Gaston, D., Monmege, B., Reynier, P.-A.: Optimal reachability in divergent weighted timed games. Research Report 1701.03716, arXiv, January 2017
18. Hansen, T.D., Ibsen-Jensen, R., Miltersen, P.B.: A faster algorithm for solving one-clock priced timed games. In: D'Argenio, P.R., Melgratti, H. (eds.) CONCUR 2013. LNCS, vol. 8052, pp. 531–545. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40184-8_37
19. Immerman, N.: Number of quantifiers is better than number of tape cells. J. Comput. Syst. Sci. **22**(3), 384–406 (1981)
20. Immerman, N.: Nondeterministic space is closed under complementation. SIAM J. Comput. **17**, 935–938 (1988)
21. Jurdziński, M., Trivedi, A.: Reachability-time games on timed automata. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 838–849. Springer, Heidelberg (2007). doi:10.1007/978-3-540-73420-8_72
22. Khachiyan, L., Boros, E., Borys, K., Elbassioni, K., Gurvich, V., Rudolf, G., Zhao, J.: On short paths interdiction problems: total and node-wise limited interdiction. Theor. Comput. Syst. **43**(2), 204–233 (2008)
23. Puri, A.: Dynamical properties of timed automata. Discrete Event Dyn. Syst. **10**(1–2), 87–113 (2000)
24. Rutkowski, M.: Two-player reachability-price games on single-clock timed automata. In Proceedings of the Ninth Workshop on Quantitative Aspects of Programming Languages (QAPL2011), vol. 57 of EPTCS, pp. 31–46 (2011)
25. Szelepcsényi, R.: The method of forced enumeration for nondeterministic automata. Acta Informatica **26**(3), 279–284 (1988)

Bounding Average-Energy Games

Patricia Bouyer[1], Piotr Hofman[1,2], Nicolas Markey[1,3], Mickael Randour[4(✉)], and Martin Zimmermann[5]

[1] LSV, CNRS & ENS Cachan, Université Paris Saclay, Cachan, France
[2] University of Warsaw, ul. Banacha 2, 02-097 Warszawa, Poland
[3] IRISA, CNRS & INRIA & U. Rennes 1, Rennes, France
[4] Computer Science Department, ULB - Université Libre de Bruxelles, Brussels, Belgium
mickael.randour@gmail.com
[5] Reactive Systems Group, Saarland University, 66123 Saarbrücken, Germany

Abstract. We consider average-energy games, where the goal is to minimize the long-run average of the accumulated energy. While several results have been obtained on these games recently, decidability of average-energy games with a lower-bound constraint on the energy level (but no upper bound) remained open; in particular, so far there was no known upper bound on the memory that is required for winning strategies.

By reducing average-energy games with lower-bounded energy to infinite-state mean-payoff games and analyzing the density of low-energy configurations, we show an almost tight doubly-exponential upper bound on the necessary memory, and prove that the winner of average-energy games with lower-bounded energy can be determined in doubly-exponential time. We also prove EXPSPACE-hardness of this problem.

Finally, we consider multi-dimensional extensions of all types of average-energy games: without bounds, with only a lower bound, and with both a lower and an upper bound on the energy. We show that the fully-bounded version is the only case to remain decidable in multiple dimensions.

1 Introduction

Quantitative two-player games of infinite duration provide a natural framework for synthesizing controllers for reactive systems with resource restrictions in an antagonistic environment (see e.g., [1,23]). In such games, player P_0 (who represents the system to be synthesized) and player P_1 (representing the antagonistic environment) construct an infinite path by moving a pebble through a graph, which describes the interaction between the system and its environment.

A full version of this paper [4] is available online: https://arxiv.org/abs/1610.07858.

P. Bouyer and N. Markey—Supported by ERC project EQualIS (308087).
M. Randour—F.R.S.-FNRS postdoctoral researcher.
M. Zimmermann—Supported by the DFG project TriCS (ZI 1516/1-1).

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 179–195, 2017.
DOI: 10.1007/978-3-662-54458-7_11

The objective, a subset of the infinite paths that encodes the controller's specification, determines the winner of such a play. Quantitative games extend this classical model by weights on edges for modeling costs, consumption, or rewards, and by a quantitative objective to encode the specification in terms of the weights.

As an example, consider the game in Fig. 1: we interpret negative weights as energy consumption and correspondingly positive weights as recharges. Then, P_0 (who moves the pebble at the circular states) can always maintain an energy level (the sum of the weights seen along a play prefix starting with energy zero) between zero and six using the following strategy: when at state s_0 with energy level at least two, go to state s_1, otherwise go to state s_2 in order to satisfy the lower bound. At state s_1, always move to s_0. It is straightforward to verify that the strategy has the desired property when starting at the initial state s_0 with initial energy zero. Note that this strategy requires memory to be implemented, as its choices depend on the current energy level and not only on the state the pebble is currently at.

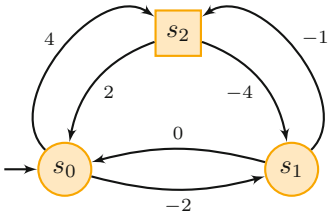


Fig. 1. Simple weighted game.

Formally, the *energy objective* requires P_0 to maintain an energy level within some (lower and/or upper) bounds, which are either given as input or existentially quantified. In the example above, P_0 has a strategy to win for the energy objective with lower bound zero and upper bound six. Energy objectives [3,11,18,19] and their combinations with parity objectives [9,11] have received significant attention in the literature.

However, a plain energy (parity) objective is sometimes not sufficient to adequately model real-life systems. For example, consider the following specification for the controller of an oil pump, based on a case study [7]: it has to keep the amount of oil in an accumulator within given bounds (an energy objective with given lower and upper bounds) while keeping the average amount of oil in the accumulator below a given threshold *in the long run*. The latter requirement reduces the wear and tear of the system, but cannot be expressed as an energy objective nor as a parity objective. Constraints on the long-run average energy level (which exactly represents the amount of oil in our example) can be specified using the *average-energy objective* [5]. As seen in this example, they are typically studied in conjunction with bounds on the energy level.

Recall the example in Fig. 1. The aforementioned strategy for P_0 guarantees a long-run average energy level, i.e., *average-energy*, of at most 11/4 (recall we want to minimize it): the outcome with worst average-energy is $\left(s_0 s_2 (s_0 s_1)^3\right)^\omega$, with energy levels $(4, 6, 4, 4, 2, 2, 0, 0)^\omega$.

The average-energy objective was first introduced by Thuijsman and Vrieze under the name *total-reward* [24] (there is an unrelated, more standard, objective called total-reward, see [5] for a discussion). Recently, the average-energy objective was independently revisited by Boros *et al.* [2] and by Bouyer *et al.* [5]. The former work studies Markov decision processes and stochastic games with

average-energy objectives. The latter studies non-stochastic games with average-energy objectives, with or without lower and upper bounds on the energy level; it determines the complexity of computing the winner and the memory requirements for winning strategies in such games. In particular, it solves games with average-energy objectives with *both* an upper and a lower bound on the energy level by a reduction to mean-payoff games: to this end, the graph is extended to track the energy level between these bounds (a losing sink for P_0 is reached if these bounds are exceeded). Thus, the bounds on the energy level are already taken care of and the average-energy objective can now be expressed as a *mean-payoff objective* [13], as the new graph encodes the current energy level in its weights. This reduction yields an exponential-time decision algorithm. Moreover, it is shown in [5] that these games are indeed EXPTIME-complete. Note that the algorithm crucially depends on the upper bound being given as part of the input, which implies that the graph of the extended game is still finite.

One problem left open in [5] concerns average-energy games *with only a lower bound* on the energy level: computing the winner is shown to be EXPTIME-hard, but it is not known whether this problem is decidable at all. Similarly, pseudo-polynomial lower bounds (i.e., lower bounds that are polynomial in the *values* of the weights, but possibly exponential in the size of their binary representations) on the necessary memory to implement a winning strategy for P_0 are given, but no upper bound is known. The major obstacle toward solving these problems is that without an upper bound on the energy, a strategy might allow arbitrarily large energy levels while still maintaining a bounded average, by enforcing long stretches with a small energy level to offset the large levels.

A step toward resolving these problems was taken by considering two variants of energy and average-energy objectives where (i) the upper bound on the energy level, or (ii) the threshold on the average energy level, is existentially quantified [21]. It turns out that these two variants are equivalent. One direction is trivial: if the energy is bounded, then the average-energy is bounded. On the other hand, if P_0 can guarantee some upper bound on the average, then he can also guarantee an upper bound on the energy level, i.e., an (existential) average-energy objective can always be satisfied with bounded energy levels. This is shown by transforming a strategy satisfying a bound on the average (but possibly allowing arbitrarily high energy levels) into one that bounds the energy by skipping parts of plays where the energy level is much higher than the threshold on the average. However, the proof is not effective: it does not yield an upper bound on the necessary energy level, just a guarantee that some bound exists. Even more so, it is still possible that the average has to increase when keeping the energy bounded. Hence, it does not answer our problem: does achieving a *given* threshold on the average-energy require unbounded energy levels and infinite memory?

Another potential approach toward solving the problem is to extend the reduction presented in [5] (which goes from average-energy games with both lower and upper bound on the energy level to mean-payoff games) to games without such an upper bound, which results in an infinite graph. This graph

can be seen as the configuration graph of a one-counter pushdown system, i.e., the stack height corresponds to the current energy level, and the average-energy objective is again transformed into a mean-payoff objective, where the weight of an edge is given by the stack height at the target of the edge. Hence, the weight function is unbounded. To the best of our knowledge, such mean-payoff games have not been studied before. However, mean-payoff games on pushdown systems with bounded weight functions are known to be undecidable [12].

Our Contribution. We develop the first algorithm for solving games with average-energy objectives and a lower bound on the energy level, and give an upper bound on the necessary memory to implement a winning strategy for P_0 in such games.

First, we present an algorithm for solving such games in doubly-exponential time (for the case of a binary encoding of the weights). The algorithm is based on the characterization of an average-energy game as a mean-payoff game on an infinite graph described above. If the average-energy of a play is bounded by the threshold t, then configurations with energy level at most t have to be visited frequently. As there are only finitely many such configurations, we obtain cycles on this play. By a more fine-grained analysis, we obtain such a cycle with an average of at most t and whose length is bounded exponentially. Finally, by analyzing strategies for reachability objectives in pushdown games, we show that P_0 can ensure that the distance between such cycles is bounded doubly-exponentially. From these properties, we obtain a doubly-exponential upper bound on the necessary energy level to ensure an average-energy of at most t. The resulting equivalent average-energy game with a lower and an upper bound can be solved in doubly-exponential time. Furthermore, if the weights and the threshold are encoded in unary (or are bounded polynomially in the number of states), then we obtain an exponential-time algorithm.

Second, from the reduction sketched above, we also obtain a doubly-exponential upper bound on the necessary memory for P_0, the first such bound. In contrast, a certain succinct one-counter game due to Hunter [16], which can easily be expressed as an average-energy game with threshold zero, shows that our bound is almost tight: in the resulting game of size n, energy level $2^{(2^{\sqrt{n}}/\sqrt{n})-1}$ is necessary to win. Again, in the case of unary encodings, we obtain an (almost) tight exponential bound on the memory requirements.

Third, we improve the lower bound on the complexity of solving average-energy games with only a lower bound on the energy level from EXPTIME to EXPSPACE by a reduction from succinct one-counter games [17].

Fourth, we show that multi-dimensional average-energy games are undecidable, both for the case without any bounds and for the case of only lower bounds. Only the case of games with both lower and upper bounds turns out to be decidable: it is shown to be both in NEXPTIME and in coNEXPTIME. This problem trivially inherits EXPTIME-hardness from the one-dimensional case.

2 Preliminaries

Graph games. We consider finite turn-based weighted games played on graphs between two players, denoted by P_0 and P_1. Such a game is a tuple $G = (S_0, S_1, E)$ where (i) S_0 and S_1 are disjoint sets of *states* belonging to P_0 and P_1, with $S = S_0 \uplus S_1$, (ii) $E \subseteq S \times [-W; W] \times S$, for some $W \in \mathbb{N}$, is a set of integer-weighted *edges*. Given an edge $e = (s, w, t) \in E$, we write $src(e)$ for the source state s of e, $tgt(e)$ for its target state t, and $w(e)$ for its weight w. We assume that for every $s \in S$, there is at least one outgoing edge $(s, w, s') \in E$.

Let $s \in S$. A *play* from s is an infinite sequence of edges $\pi = (e_i)_{1 \leq i}$ such that $src(e_1) = s$ and $tgt(e_i) = src(e_{i+1})$ for all $i \geq 1$. A play induces a corresponding sequence of states, denoted $\hat{\pi} = (s_j)_{0 \leq j}$, such that for any e_i, $i \geq 1$, in π, $s_{i-1} = src(e_i)$ and $s_i = tgt(e_i)$. We write $first(\pi) = s_0$ for its initial state (here, s). A play *prefix* from s is a finite sequence of edges $\rho = (e_i)_{1 \leq i \leq k}$ following the same rules and notations. We additionally write $last(\rho) = s_k = tgt(e_k)$ for its last state. We let ϵ_s (or ϵ when s is clear from the context) denote the empty play prefix from s, with $last(\epsilon_s) = first(\epsilon_s) = s$. A non-empty prefix ρ such that $last(\rho) = first(\rho)$ is called a *cycle*. The length of a prefix $\rho = (e_i)_{1 \leq i \leq k}$ is its number of edges, i.e., $\ell(\rho) = k$. For a play π, $\ell(\pi) = \infty$. Given a prefix ρ and a play (or prefix) π with $last(\rho) = first(\pi)$, the concatenation between ρ and π is denoted by $\rho \cdot \pi$.

For a play $\pi = (e_i)_{1 \leq i}$ and $1 \leq j \leq k$, we write $\pi_{[j,k]}$ to denote the finite sequence $(e_i)_{j \leq i \leq k}$, which is a prefix from $src(e_j)$; we write $\pi_{\leq k}$ for $\pi_{[1,k]}$. For any $i \geq 1$ and $j \geq 0$, we write π_i for edge e_i and $\hat{\pi}_j$ for state s_j. Similar notations are used for prefixes ρ, with all indices bounded by $\ell(\rho)$.

The set of all plays in G from a state s is denoted by $Plays(G, s)$, and the set of all such prefixes is denoted by $Prefs(G, s)$. We write $Plays(G)$ and $Prefs(G)$ for the unions of those sets over all states. We say that a prefix $\rho \in Prefs(G)$ belongs to P_i, for $i \in \{0, 1\}$, if $last(\rho) \in S_i$. The set of prefixes that belong to P_i is denoted by $Prefs_i(G)$, and we define $Prefs_i(G, s) = Prefs_i(G) \cap Prefs(G, s)$.

Payoffs. Given a non-empty prefix $\rho = (e_i)_{1 \leq i \leq n}$, we define the following payoffs:

- its *energy level* as $\mathsf{EL}(\rho) = \sum_{i=1}^{n} w(e_i)$;
- its *mean-payoff* as $\mathsf{MP}(\rho) = \frac{1}{n} \sum_{i=1}^{n} w(e_i) = \frac{1}{n}\mathsf{EL}(\rho)$;
- its *average-energy* as $\mathsf{AE}(\rho) = \frac{1}{n} \sum_{i=1}^{n} \mathsf{EL}(\rho_{\leq i})$.

These definitions are extended to plays by taking the upper limit of the respective functions applied to the sequence of prefixes of the plays, e.g.,

$$\overline{\mathsf{AE}}(\pi) = \limsup_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} \mathsf{EL}(\pi_{\leq i}).$$

Example 1. We illustrate those definitions on a small example depicted in Fig. 2: it displays two small (1-player, deterministic) weighted games, together with the evolution of the energy level and average-energy along their unique play. As noted in [5], the *average-energy* can help in discriminating plays that have

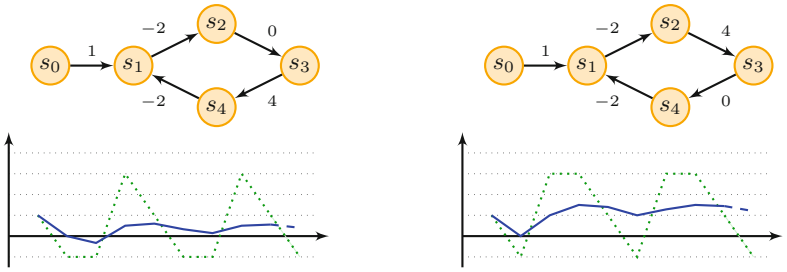


Fig. 2. Two plays with identical mean-payoffs and total-payoffs. The left one has average-energy 1/2, in contrast to 3/2 for the right one. Green (dotted) and blue curves respectively represent the energy level and the average-energy over prefixes. (Color figure online)

identical *total-payoffs* (i.e., the limits of high and low points in the sequence of energy levels), in the same way that total-payoff can discriminate between plays having the same *mean-payoff*. Indeed, in our example, both plays have mean-payoff equal to zero and supremum (resp. infimum) total-payoff equal to three (resp. -1), but they end up having different averages: the average-energy is $1/2$ for the left play, while it is $3/2$ for the right one.

Strategies. A *strategy* for P_i, with $i \in \{0,1\}$, from a state s is a function $\sigma_i \colon \mathit{Prefs}_i(G,s) \to E$ satisfying $\mathit{src}(\sigma_i(\rho)) = \mathit{last}(\rho)$ for all $\rho \in \mathit{Prefs}_i(G,s)$. We denote by $\mathit{Strats}_i(G,s)$, the set of strategies for P_i from state s. We drop G and s when they are clear from the context.

A play $\pi = (e_j)_{1 \le j}$ from s is called an *outcome* of strategy σ_i of P_i if, for all $k \ge 0$ where $\pi_{\le k} \in \mathit{Prefs}_i(G,s)$, we have $\sigma_i(\pi_{\le k}) = e_{k+1}$. Given a state $s \in S$ and strategies σ_0 and σ_1 from s for both players, we denote by $\mathit{Out}(s,\sigma_0,\sigma_1)$ the unique play that starts in s and is an outcome of both σ_0 and σ_1. When fixing the strategy of only P_i, we denote the set of outcomes by

$$\mathit{Out}(s,\sigma_i) = \{\mathit{Out}(s,\sigma_0,\sigma_1) \mid \sigma_{1-i} \in \mathit{Strats}_{1-i}(G,s)\}.$$

Objectives. An objective in G is a set $\mathcal{W} \subseteq \mathit{Plays}(G)$. Given a game G, an initial state s_{init}, and an objective $\mathcal{W}$, a strategy $\sigma_0 \in \mathit{Strats}_0$ is winning for P_0 if $\mathit{Out}(s_{\mathsf{init}},\sigma_0) \subseteq \mathcal{W}$. We consider the following objectives for P_0:

- The **lower-bounded energy** objective $\mathit{Energy}_L = \{\pi \in \mathit{Plays}(G) \mid \forall\, n \ge 1, \mathsf{EL}(\pi_{\le n}) \ge 0\}$ requires a non-negative energy level at all times.[1]
- Given an upper bound $U \in \mathbb{N}$, the **lower- and upper-bounded energy** objective $\mathit{Energy}_{LU}(U) = \{\pi \in \mathit{Plays}(G) \mid \forall\, n \ge 1,\ \mathsf{EL}(\pi_{\le n}) \in [0,U]\}$ requires that the energy always remains non-negative and below the upper bound U along a play.

[1] For the sake of readability, we assume the initial credit to be zero for energy objectives throughout this paper. Still, our techniques can easily be generalized to an arbitrary initial credit $c_{\mathsf{init}} \in \mathbb{N}$.

Table 1. Complexity of deciding the winner and memory requirements for quantitative games: *MP* stands for mean-payoff, *EGL* (resp. *EGLU*) for lower-bounded (resp. lower- and upper-bounded) energy, *AE* for average-energy, *AEL* (resp. *AELU*) for average-energy under a lower bound (resp. and upper bound) on the energy, c. for complete, e. for easy, and h. for hard. All memory bounds are tight (except for *AEL*).

Objective	1-player	2-player	Memory
MP	PTIME[20]	NP ∩ coNP [27]	Memoryless [13]
EGL	PTIME [3]	NP ∩ coNP [3,8]	Memoryless [8]
EGLU	PSPACE-c. [14]	EXPTIME-c. [3]	Exponential [5]
AE	PTIME [5]	NP ∩ coNP[5]	Memoryless [5]
AELU	PSPACE-c. [5]	EXPTIME-c. [5]	Exponential [5]
AEL	PSPACE-e./NP-h. [5]	EXPSPACE-h.	At least super-exp.
		2-EXPTIME-e.	At most doubly-exp.

– Given a threshold $t \in \mathbb{Q}$, the **mean-payoff** objective $\mathit{MeanPayoff}(t) = \{\pi \in \mathit{Plays}(G) \mid \overline{\mathsf{MP}}(\pi) \leq t\}$ requires that the mean-payoff is at most t.
– Given a threshold $t \in \mathbb{Q}$, the **average-energy** objective $\mathit{AvgEnergy}(t) = \{\pi \in \mathit{Plays}(G) \mid \overline{\mathsf{AE}}(\pi) \leq t\}$ requires that the average-energy is at most t.

For the *MeanPayoff* and *AvgEnergy* objectives, P_0 aims to *minimize* the payoff.

Decision problem. In this paper, we focus on weighted games with a combination of energy and average-energy objectives, by a detour via mean-payoff objectives. The exact problem we tackle is named the *AEL threshold problem* and is defined as follows: given a finite game G, an initial state $s_{\mathsf{init}} \in S$, and a threshold $t \in \mathbb{Q}$ given as a fraction $\frac{t_1}{t_2}$ with t_1 and t_2 two natural numbers given in binary, decide whether P_0 has a winning strategy from s_{init} for the objective $\mathit{AvgEnergy}_L(t) = \mathit{Energy}_L \cap \mathit{AvgEnergy}(t)$. As for the threshold, we consider a binary encoding of the weights in G: we thus study the complexity of the problem with regard to the length of the input's binary encoding (i.e., the number of bits used to represent the graph and the numbers involved).

Variants of this problem involving the above-mentioned payoff functions, and combinations thereof, had been previously investigated, see Table 1 for a summary of the results. In this paper, we focus on the remaining case, namely 2-player games with *AEL* objectives, for which decidability was not known, and proving the computational- and memory complexities given in the corresponding cells of the table.

3 Equivalence with an Infinite-State Mean-Payoff Game

Let $G = (S_0, S_1, E)$ be a finite weighted game, $s_{\mathsf{init}} \in S$ be an initial state, and $t \in \mathbb{Q}$ be a threshold. We define its *expanded infinite-state weighted game* as $G' = (\Gamma_0, \Gamma_1, \Delta)$ defined by

– $\Gamma_0 = S_0 \times \mathbb{N}$, and $\Gamma_1 = S_1 \times \mathbb{N} \uplus \{\bot\}$ (where $\bot$ is a fresh sink state that does not belong to G); then $\Gamma = \Gamma_0 \uplus \Gamma_1$ is the global set of states;
– Δ is composed of the following edges:
  - a transition $((s,c),c',(s',c')) \in \Delta$ whenever there is $(s,w,s') \in E$ with $c' = c + w \geq 0$;
  - a transition $((s,c),\lceil t \rceil + 1,\bot) \in \Delta$ whenever there is $(s,w,s') \in E$ such that $c + w < 0$;
  - finally, a transition $(\bot,\lceil t \rceil + 1,\bot) \in \Delta$.

In this expanded game, elements of Γ are called *configurations*, and the initial configuration is set to $(s_{\mathsf{init}},0)$.

Lemma 1. *Player P_0 has a winning strategy in G from state s_{init} for the objective $AvgEnergy_L(t)$ if, and only if, he has a winning strategy in G' from configuration $(s_{\mathsf{init}},0)$ for the objective MeanPayoff(t).*

For the rest of this paper, we fix a weighted game $G = (S_0,S_1,E)$ and a threshold $t \in \mathbb{Q}$, and work on the corresponding expanded weighted game $G' = (\Gamma_0,\Gamma_1,\Delta)$. We write $t = \frac{t_1}{t_2} = \lfloor t \rfloor + \frac{t'}{t_2}$, where $t_1,t_2,t' \in \mathbb{N}$ (recall they are given in binary), and $0 \leq t' < t_2$, and $\lfloor t \rfloor$ stands for the integral part of t. We also let $\widetilde{t} = \lfloor t \rfloor + 1 - t = 1 - \frac{t'}{t_2}$. Hence $\widetilde{t} = 1$ indicates that t is an integer. For a given threshold $t \in \mathbb{Q}$, we consider $\Gamma^{\leq t} = \{(s,c) \in \Gamma \mid c \leq t\}$, i.e., the set of configurations below the threshold.

Note that G' can be interpreted as a one-counter pushdown mean-payoff game with an unbounded weight function. While it is well-known how to solve mean-payoff games over *finite* arenas, not much is known for infinite arenas (see Sect. 1). However, our game has a special structure that we will exploit to obtain an algorithm. Roughly, our approach consists in transforming the $AvgEnergy_L(t)$ objective into an equivalent $AvgEnergy_{LU}(t,U) = Energy_{LU}(U) \cap AvgEnergy(t)$ objective, where (the value of) U is doubly-exponential in the input by analyzing plays and strategies in G'. In other terms, we show that any winning strategy for $AvgEnergy_L(t)$ can be transformed into another winning strategy along which the energy level remains upper-bounded by U.

The proof is two-fold: we first show (in Sect. 4) that we can bound the energy level for the special case where the objective consists in reaching a finite set of configurations of the game (with only a lower bound on the energy level). This is achieved by a detour to pushdown games: while there are known algorithms for solving reachability pushdown games, to the best of our knowledge, there are no (explicit) results bounding the maximal stack height.

As a second step (in Sect. 5), we identify *good cycles* in winning outcomes, and prove that they can be shown to have bounded length. The initial configurations of those cycles will then be the targets of the reachability games above. Combining these two results yields the desired upper bound on the energy levels.

4 Bounding One-Counter Reachability Games

We focus here on a reachability objective in G', where the target set is a subset $\Gamma' \subseteq \Gamma^{\leq t}$: we aim at bounding the maximal energy level that needs to be visited with a winning strategy.

The game G' is a particular case of a pushdown game [26]. Hence we use results on pushdown games, and build a new winning strategy, in which we will be able to bound the energy level at every visited configuration. Note that the bound M' in the following lemma is doubly-exponential, as we encode W, the largest absolute weight in G, and the threshold t, in binary. The proof of the next lemma is based on the reformulation of the algorithm from [26] made in [15].

Lemma 2. *Fix $M \in \mathbb{N}$. There exists $M' = 2^{\mathcal{O}(M+|S|+|E|\cdot W+|S|\cdot(\lceil t \rceil+1))}$ such that for every $\gamma = (s, c)$ with $c \leq M$ and for every $\Gamma' \subseteq \Gamma^{\leq t}$, if there is a strategy for P_0 to reach Γ' from γ in G', then there is also a strategy which ensures reaching Γ' from γ without exceeding energy level M'.*

5 A Doubly-Exponential Time Algorithm

Let $\widetilde{\Gamma} \subseteq \Gamma$ be a set of configurations of G', and ρ be a play prefix. We define

$$\mathbf{d}(\widetilde{\Gamma}, \rho) = \frac{|\{1 \leq i \leq \ell(\rho) \mid \hat{\rho}_i \in \widetilde{\Gamma}\}|}{\ell(\rho)},$$

which denotes the proportion (or *density*) of configurations belonging to $\widetilde{\Gamma}$ along ρ. Observe that the initial configuration $\hat{\rho}_0$ is not taken into account: this is because $\mathbf{d}(\widetilde{\Gamma}, \rho)$ will be strongly linked to the mean-payoff, as we now explain.

5.1 Analyzing Winning Plays

In this section, we analyze winning plays in G', and prove that they must contain a cycle that is "short enough" and has mean-payoff less than or equal to t.

To achieve this, we observe that if the mean-payoff of a play π is less than t, then there must exist a configuration $\gamma \in \Gamma^{\leq t}$ that appears frequently enough along π. Applying a sequence of elementary arguments, we can even give a uniform lower bound on the density of γ along arbitrarily far and arbitrarily long segments of π. More precisely, we show:

Lemma 3. *Let π be a play in G' from $(s_{\mathsf{init}}, 0)$ with $\overline{\mathsf{MP}}(\pi) \leq t$. There exists $\gamma \in \Gamma^{\leq t}$ such that for any $n \in \mathbb{N}$, there are infinitely many positions $n' \geq n$ for which*

$$\mathbf{d}(\{\gamma\}, \pi_{[n,n']}) \geq \frac{\widetilde{t}}{4(t+1)^2|S|}.$$

Next we say that a cycle of G' is *good* whenever it starts in $\Gamma^{\leq t}$ and its mean-payoff is bounded by t. Since γ appears frequently along π, we shall argue that it is not possible that all cycles along π delimited by γ are bad (i.e., not good), otherwise the global mean-payoff of π could not be bounded by t.

Hence we obtain that π contains a good cycle delimited by γ. It remains to argue that such a minimal-length good cycle (i.e., with no good nested sub-cycle) cannot be too long. We write $\mathcal{C}$ for a minimal-length good cycle delimited by γ. This part of the proof appeals to a second density argument for γ along $\mathcal{C}$: since $\mathcal{C}$ does not contain good sub-cycles, it cannot contain too many sub-cycles at all. Letting $N = 8t_1t_2(t+1)^3|S|^2$, we prove:

Proposition 4. *Let π be a play in G' from $(s_{\mathsf{init}}, 0)$ with $\overline{\mathsf{MP}}(\pi) \leq t$. Then there exist $1 \leq i \leq j$ such that $\pi_{[i,j]}$ is a good cycle of length at most N.*

5.2 Strategies Described by Finite Trees

So far, we have proven that P_0 should "target" short good cycles. However in a two-player context, P_1 might prevent P_0 from doing so. We therefore need to consider the *branching* behaviour of the game, and not only the linear point-of-view given by a play. Toward that aim, we represent strategies (of P_0) as *strategy trees*, and use them to bound the amount of memory and the counter values needed to win in G'.

We consider labelled trees with backward edges $\mathcal{T} = (\mathcal{N}, \mathcal{E}, \lambda, \dashrightarrow)$, where $\mathcal{N}$ is a finite set of nodes, $\lambda \colon \mathcal{N} \to S \times \mathbb{N}$ (each node is labelled with a configuration of the game G'), and $\dashrightarrow \subseteq \mathcal{N} \times \mathcal{N}$. We assume $\mathcal{T}$ has at least two nodes. The relation $\mathcal{E}$ is the successor relation between nodes. A node with no $\mathcal{E}$-successor is called a *leaf*; other nodes are called *internal nodes*. The root of $\mathcal{T}$, denoted by $\mathbf{n}_{\mathsf{root}}$, is the only node having no predecessor. The relation $\dashrightarrow$ is an extra relation between nodes that will become clear later.

For such a tree to represent a strategy, we require that each internal node $\mathbf{n}$ that is labelled by a P_0-configuration (s, c) has only one successor $\mathbf{n}'$, with $\lambda(\mathbf{n}') = (s', c')$ such that there is a transition $((s, c), c', (s', c'))$ in the game G'; we require that each internal node $\mathbf{n}$ that is labelled with a P_1-state (s, c) has exactly one successor per transition $((s, c), c', (s', c'))$ in G', each successor being labelled with its associated (s', c'). Finally, we require that each leaf $\mathbf{n}$ of $\mathcal{T}$ has a (strict) ancestor node $\mathbf{n}'$ such that $\lambda(\mathbf{n}') = \lambda(\mathbf{n})$. The relation $\dashrightarrow$ will serve witnessing that property. So we assume that for every leaf $\mathbf{n}$, there is a unique ancestor node $\mathbf{n}'$ such that $\mathbf{n} \dashrightarrow \mathbf{n}'$; furthermore it should be the case that $\lambda(\mathbf{n}') = \lambda(\mathbf{n})$. Under all these constraints, $\mathcal{T}$ is called a *strategy tree*. It basically represents a (finite) memory structure for a strategy, as we now explain.

Let $\mathcal{T} = (\mathcal{N}, \mathcal{E}, \lambda, \dashrightarrow)$ be strategy tree for G'. We define $\mathcal{G}_{\mathcal{T}} = (\mathcal{N}, \mathcal{E}')$, a directed graph obtained from $\mathcal{T}$ by adding extra edges $(\mathbf{n}, \mathbf{n}'')$ for each leaf $\mathbf{n}$ and node $\mathbf{n}''$ for which there exists another node $\mathbf{n}'$ satisfying $\mathbf{n} \dashrightarrow \mathbf{n}'$ and $(\mathbf{n}', \mathbf{n}'') \in \mathcal{E}$. We refer to these extra edges as *back-edges*. One may notice that for any $(\mathbf{n}, \mathbf{n}') \in \mathcal{E}'$ there is an edge from $\lambda(\mathbf{n})$ to $\lambda(\mathbf{n}')$ in G'. Given two nodes

$\mathbf{n}$ and $\mathbf{n}'$ such that $\mathbf{n}'$ is an ancestor of $\mathbf{n}$ in $\mathcal{T}$, we write $[\mathbf{n}' \rightsquigarrow \mathbf{n}]$ for the play prefix from $\mathbf{n}'$ to $\mathbf{n}$ (inclusive) using only transitions from $\mathcal{E}$.

Now, we associate with any prefix ρ in $\mathcal{G}_\mathcal{T}$ from $\mathbf{n}_{\mathsf{root}}$ a prefix $\overline{\rho}$ in G' from $\lambda(\mathbf{n}_{\mathsf{root}}) = (s_{\mathsf{root}}, c_{\mathsf{root}})$ such that $last(\overline{\rho}) = \lambda(last(\rho))$. The construction is inductive:

- with the empty prefix in $\mathcal{G}_\mathcal{T}$ we associate the one in G': $\overline{\epsilon_{\mathbf{n}_{\mathsf{root}}}} = \epsilon_{(s_{\mathsf{root}}, c_{\mathsf{root}})}$,
- if $\rho = \rho' \cdot (\mathbf{n}', \mathbf{n})$ with $(\mathbf{n}', \mathbf{n}) \in \mathcal{E}'$, writing $(s', c') = \lambda(\mathbf{n}')$ and $(s, c) = \lambda(\mathbf{n})$, then $\overline{\rho} = \overline{\rho'} \cdot ((s', c'), c, (s, c))$ (which by construction is indeed a prefix in G').

We now explain how $\mathcal{G}_\mathcal{T}$ corresponds to a strategy in G': for any prefix ρ in $\mathcal{G}_\mathcal{T}$, if $\lambda(last(\rho)) = (s, c) \in \Gamma_0$, then $last(\rho)$ has a unique successor $\mathbf{n}'$ in $\mathcal{G}_\mathcal{T}$, and, writing $(s', c') = \lambda(\mathbf{n}')$, we define $\sigma_\mathcal{T}(\overline{\rho}) = ((s, c), c', (s', c'))$: $\sigma_\mathcal{T}$ is a (partially defined) strategy in G'. The following lemma states that $\mathcal{G}_\mathcal{T}$ actually represents the outcomes of the well-defined strategy $\sigma_\mathcal{T}$ from $\lambda(\mathbf{n}_{\mathsf{root}})$ in G':

Lemma 5. *Let μ be a prefix in G' from $(s_{\mathsf{root}}, c_{\mathsf{root}})$. Assume that for every $i \leq \ell(\mu)$ such that $last(\mu_{\leq i}) \in \Gamma_0$, the function $\sigma_\mathcal{T}$ is defined on $\mu_{\leq i}$ and $\mu_{\leq i+1} = \mu_{\leq i} \cdot \sigma_\mathcal{T}(\mu_{\leq i})$. Then there exists a unique prefix ρ in $\mathcal{G}_\mathcal{T}$ such that $\mu = \overline{\rho}$, Moreover, if $last(\mu) \in \Gamma_0$, then $\sigma_\mathcal{T}(\mu)$ is defined.*

We now give conditions for $\sigma_\mathcal{T}$ to be a winning strategy from $(s_{\mathsf{root}}, c_{\mathsf{root}})$ in G'. With a finite outcome $\mu = \overline{\rho}$ of $\sigma_\mathcal{T}$ from $(s_{\mathsf{root}}, c_{\mathsf{root}})$, we associate a sequence $decomp_\mathcal{T}(\mu)$ of cycles in G', defined inductively as follows:

- $decomp_\mathcal{T}(\epsilon_{(s_{\mathsf{root}}, c_{\mathsf{root}})})$ is empty;
- if $\rho = \rho' \cdot (\mathbf{n}', \mathbf{n})$ and $\mathbf{n}$ is not a leaf of $\mathcal{T}$, then $decomp_\mathcal{T}(\overline{\rho}) = decomp_\mathcal{T}(\overline{\rho'})$;
- if $\rho = \rho' \cdot (\mathbf{n}', \mathbf{n})$ and $\mathbf{n}$ is a leaf of $\mathcal{T}$, we let $\mathbf{n}''$ be such that $\mathbf{n} \dashrightarrow \mathbf{n}''$; the prefix $[\mathbf{n}'' \rightsquigarrow \mathbf{n}]$ in $\mathcal{T}$ corresponds to a cycle C in G', and we let $decomp_\mathcal{T}(\overline{\rho}) = decomp_\mathcal{T}(\overline{\rho'}) \cdot C$.

The sequence $decomp_\mathcal{T}(\overline{\rho})$ hence contains all full cycles (closed at leaves) encountered while reading ρ in $\mathcal{T}$: hence it comprises all edges of ρ except a prefix starting at $\mathbf{n}_{\mathsf{root}}$ and a suffix since the last back-edge has been taken. It is not hard to see that those can actually be concatenated. By induction, we can easily show:

Proposition 6. *Let μ be a non-empty finite outcome of $\sigma_\mathcal{T}$ from $(s_{\mathsf{root}}, c_{\mathsf{root}})$ in G'. Write $decomp_\mathcal{T}(\mu) = C_0 \cdot C_1 \cdot \ldots \cdot C_h$ (where each C_i is a cycle). Let ρ be the prefix in $\mathcal{G}_\mathcal{T}$ such that $\mu = \overline{\rho}$, $\mathbf{n} = last(\rho)$, and $\nu = [\mathbf{n}_{\mathsf{root}} \rightsquigarrow \mathbf{n}]$. Write $(s_j, c_j) = \lambda(\hat{\nu}_j)$. Then:*

$$\mathsf{MP}(\mu) = \frac{\sum_{i=0}^{h} \mathsf{MP}(C_i) \cdot \ell(C_i) + \sum_{j=1}^{\ell(\nu)} c_j}{\ell(\mu)}$$

We say that a tree is *good* if, for every $\mathbf{n} \dashrightarrow \mathbf{n}'$ in $\mathcal{T}$, writing $\rho = [\mathbf{n}' \rightsquigarrow \mathbf{n}]$ and letting $\lambda(\hat{\rho}_j) = (s_j, c_j)$, it holds $\frac{\sum_{j=1}^{\ell(\rho)} c_j}{\ell(\rho)} \leq t$.

Proposition 7. *If $\mathcal{T}$ is a finite good strategy tree, then $\sigma_{\mathcal{T}}$ is a winning strategy from $(s_{\mathsf{root}}, c_{\mathsf{root}})$ in G'.*

Note that $\mathcal{T}$ can be interpreted as a *finite memory structure* for strategy $\sigma_{\mathcal{T}}$: to know which move is given by $\sigma_{\mathcal{T}}$, it is sufficient to move a token in tree $\mathcal{T}$, going down in the tree, and following back-edges when stuck at leaves.

5.3 Analyzing Winning Strategies

We proved in the previous section that the finite-memory strategy associated with a finite good strategy tree is winning. In this section, we first show the converse direction, proving that from a winning strategy, we can obtain a finite good tree. In that tree, backward edges correspond to (short) good cycles. We then use the result of Sect. 4 for showing that those parts of the tree that do not belong to a segment $[\mathbf{n} \rightsquigarrow \mathbf{n}']$ with $\mathbf{n}' \dashrightarrow \mathbf{n}$ can be replaced with other substrategies in which the counter value is uniformly bounded. That way, we show that if there is a winning strategy for our games, then there is one where the counter value is uniformly bounded along all outcomes. This will allow to apply algorithms for solving games with average-energy payoff under lower- and upper-bound constraints [5].

Fix a winning strategy σ for P_0 from (s_0, c_0) in G'. We let

$$\begin{aligned}\mathsf{goodpref}(\sigma) = \{\pi_{\leq n} \mid & \pi \in \mathit{Out}((s_0,c_0),\sigma), \text{ there is } m < n \text{ s.t.} \\ & \pi_{[m+1,n]} \text{ is a good cycle with no good strict sub-cycle} \\ & \text{and } \pi_{\leq n-1} \text{does not contain any good cycle}\}\end{aligned}$$

and for every $\pi_{\leq n} \in \mathsf{goodpref}(\sigma)$, we define $\mathit{back}(\pi_{\leq n})$ as $\pi_{\leq m}$ such that $\pi_{[m+1,n]}$ is a good cycle with no good strict sub-cycle.

We build a strategy tree $\mathcal{T}_\sigma$ as follows:

- the nodes of $\mathcal{T}_\sigma$ are all the prefixes of the finite plays in $\mathsf{goodpref}(\sigma)$; the edges relate each prefix of length k to its extensions of length $k+1$. For a node $\mathbf{n}$ corresponding to prefix $\rho_{\leq k}$ (with $\rho \in \mathsf{goodpref}(\sigma)$), we let $\lambda(\mathbf{n}) = \mathit{last}(\rho_{\leq k})$; we let $\lambda(\mathbf{n}_{\mathsf{root}}) = (s_0, c_0)$. The leaves of $\mathcal{T}_\sigma$ then correspond to the play prefixes that are in $\mathsf{goodpref}(\sigma)$.
- backward edges in $\mathcal{T}_\sigma$ are defined by noticing that each leaf $\mathbf{n}$ of $\mathcal{T}_\sigma$ corresponds to a finite path $\pi_{\leq n}$ in $\mathsf{goodpref}(\sigma)$, so that the prefix $\pi_{\leq m} = \mathit{back}(\pi_{\leq n})$ is associated with some node $\mathbf{m}$ of $\mathcal{T}_\sigma$ such that $\pi_{[m+1,n]}$ is a good cycle. This implies $\lambda(\pi_{\leq n}) = \lambda(\pi_{\leq m})$, and we define $\mathbf{n} \dashrightarrow \mathbf{m}$. This way, $\mathcal{T}_\sigma$ is a strategy tree as defined in Sect. 5.2.

Lemma 8. *Tree $\mathcal{T}_\sigma$ is a finite good strategy tree.*

Applying Proposition 7, we immediately get:

Corollary 9. *Strategy $\sigma_{\mathcal{T}_\sigma}$ is a winning strategy from (s_0, c_0).*

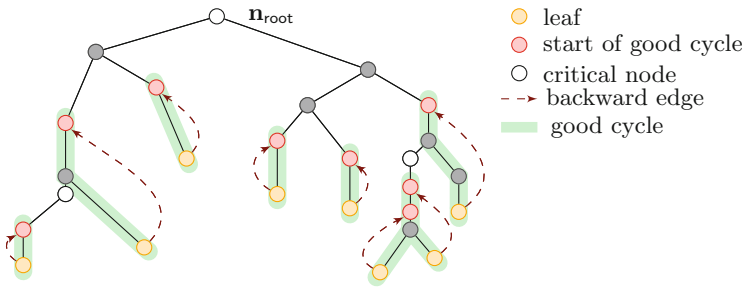


Fig. 3. Example of a finite strategy tree, with backward edges and critical nodes.

Let $\mathbf{n} \dashrightarrow \mathbf{n}'$ be two related nodes in $\mathcal{T}_\sigma$. We say that a node $\mathbf{n}''$ is *just below* $[\mathbf{n}' \rightsquigarrow \mathbf{n}]$ in $\mathcal{T}_\sigma$ whenever its predecessor appears along $[\mathbf{n}' \rightsquigarrow \mathbf{n}]$, but node $\mathbf{n}''$ itself does not appear along any path $[\mathbf{n_1} \rightsquigarrow \mathbf{n_2}]$ for which $\mathbf{n_2} \dashrightarrow \mathbf{n_1}$. Such nodes, together with the root of the tree, are called the *critical* nodes (see Fig. 3).

Lemma 10. *If* $\mathbf{n}$ *is a critical node in* $\mathcal{T}_\sigma$*, then writing* $\lambda(\mathbf{n}) = (s, c)$*, we have that* $c \leq t + W \cdot (N + 1)$.

Given a critical node $\mathbf{n}$, we define

$$\mathit{target}(\mathbf{n}) = \{\mathbf{n}' \text{ in subtree of } \mathbf{n} \mid \text{there exists } \mathbf{n}'' \text{ such that } \mathbf{n}'' \dashrightarrow \mathbf{n}' \text{ and } [\mathbf{n} \rightsquigarrow \mathbf{n}'] \text{ contains no other such node}\}.$$

Looking again at Fig. 3, the targets of a critical node are the start nodes of the good cycles that are closest to that critical node. In particular, for the rightmost critical node on Fig. 3, there are two candidate target nodes (because there are two overlapping good cycles), but only the topmost one is a target.

For every critical node $\mathbf{n}$, we apply Lemma 2 with $\gamma = \lambda(\mathbf{n})$ and $\Gamma'_\gamma = \{\gamma' = \lambda(\mathbf{n}') \mid \mathbf{n}' \in \mathit{target}(\mathbf{n})\}$, setting $M = t + W \cdot (N + 1)$. We write $\sigma_\mathbf{n}$ for the corresponding strategy: applying $\sigma_\mathbf{n}$ from $\lambda(\mathbf{n})$, player P_0 will reach some configuration (s', c') such that there is a node $\mathbf{n}' \in \mathit{target}(\mathbf{n})$ with $\lambda(\mathbf{n}') = (s', c')$.

Now, for any node $\mathbf{n}'$ that is the target of a backward edge $\mathbf{n} \dashrightarrow \mathbf{n}'$, but whose immediate predecessor does not belong to any segment $[\mathbf{n_1} \rightsquigarrow \mathbf{n_2}]$ with $\mathbf{n_2} \dashrightarrow \mathbf{n_1}$, we define strategy $\sigma_{[\mathbf{n}']}$ which follows good cycles as much as possible; when a leaf $\mathbf{m}$ is reached, the strategy replays similarly as from the equivalent node $\mathbf{m}'$ for which $\mathbf{m} \dashrightarrow \mathbf{m}'$. If, while playing that strategy, the play ever leaves a good cycle (due to a move of player P_1), then it reaches a critical node $\mathbf{n}''$. From that node, we will apply strategy $\sigma_{\mathbf{n}''}$ as defined above, and iterate like this.

This defines a strategy σ'. Applying Lemma 2 to strategies $\sigma_\mathbf{n}$ when $\mathbf{n}$ is critical, and the previous analysis of good cycles, we get the following doubly-exponential bound on the counter value (which is only exponential in case constants W, t_1, and t_2 are encoded in unary):

Proposition 11. *Strategy σ' is a winning strategy from (s_0, c_0), and all visited configurations (s, c) when applying σ' are such that $c \leq M'$ with*

$$M' = 2^{\mathcal{O}(t+W\cdot(8t_1t_2(t+1)^3|S|^2+1)+|S|+|E|\cdot W+|S|\cdot(\lceil t\rceil+1))}.$$

5.4 Conclusion

Gathering everything we have done above, we get the following equivalence.

Proposition 12. *Player P_0 has a winning strategy in game G from s_{init} for the objective $AvgEnergy_L(t)$ if, and only if, he has a wining strategy in G from s_{init} for the objective $AvgEnergy_{LU}(t, U) = Energy_{LU}(U) \cap AvgEnergy(t)$, where $U = M'$ is the bound from Proposition 11.*

Hence we can use the algorithm for games with objectives $AvgEnergy_{LU}(t, U)$ in [5], which is polynomial in $|S|$, $|E|$, t, and U (hence pseudo-polynomial only). Having in mind that the upper bound U is doubly-exponential, we can deduce our main decidability result. The memory required is also a consequence of [5].

Theorem 13. *The AEL threshold problem is in* 2-EXPTIME. *Furthermore doubly-exponential memory is sufficient to win (for player P_0).*

We could not prove a matching lower-bound, but relying on [17], we can prove EXPSPACE-hardness:

Theorem 14. *The AEL threshold problem is* EXPSPACE*-hard, even for the fixed threshold zero.*

In [16], a super-exponential lower bound is given for the required memory to win a succinct one-counter game. While the model of games is not exactly the same, the actual family of games witnessing that lower bound on the memory happens to be usable as well for the AEL threshold problem (with threshold zero). The reduction is similar to the one in the proof of Theorem 14. This yields a lower bound on the required memory to win games with $AvgEnergy_L(t)$ objectives which is $2^{(2^{\sqrt{n}}/\sqrt{n})-1}$.

For unary encodings or small weights we get better results from our technique:

Corollary 15. *The AEL threshold problem is in* EXPTIME *and exponential memory is sufficient to win (for player P_0), if the weights and the threshold are encoded in unary or polynomial in the size of the graph.*

6 Multi-dimensional Average-Energy Games

We now turn to a more general class of games where integer weights on the edges are replaced by *vectors of integer weights*, representing changes in different quantitative aspects. That is, for a game $G = (S_0, S_1, E)$ of dimension $k \geq 1$, we now have $E \subseteq S \times [-W, W]^k \times S$ for $W \in \mathbb{N}$. Multi-dimensional games

have recently gained interest as a powerful model to reason about interplays and trade-offs between different resources; and multi-dimensional versions of many classical objectives have been considered in the literature: e.g., mean-payoff [11,25], energy [11,19,25], or total-payoff [10]. We consider the natural extensions of threshold problems in the multi-dimensional setting: we take the zero vector in $\mathbb{N}^k$ as lower bound for the energy, a vector $U \in \mathbb{N}^k$ as upper bound, a vector $t \in \mathbb{Q}^k$ as threshold for the average-energy, and the payoff functions are defined using component-wise limits. That is, we essentially take the *conjunction* of our objectives for all dimensions. We quickly review the situation for the three types of average-energy objectives.

Average-energy games (without energy bounds). In the one-dimensional version of such games, memoryless strategies suffice for both players and the threshold problem is in $\mathsf{NP} \cap \mathsf{coNP}$ [5]. We prove here that already for games with three dimensions, the threshold problem is undecidable, based on a reduction from two-dimensional robot games [22]. Decidability for average-energy games with two dimensions remains open.

Theorem 16. *The threshold problem for average-energy games with three or more dimensions is undecidable. That is, given a finite k-dimensional game $G = (S_0, S_1, E)$, for $k \geq 3$, an initial state $s_{\mathsf{init}} \in S$, and a threshold $t \in \mathbb{Q}^k$, determining whether P_0 has a winning strategy from s_{init} for objective $AvgEnergy(t)$ is undecidable.*

Average-energy games with lower and upper bounds. One-dimensional versions of those games were proved to be EXPTIME-complete in [5]. The algorithm consists in reducing (in two steps) the original game to a mean-payoff game on an expanded graph of pseudo-polynomial size (polynomial in the original game but also in the upper bound $U \in \mathbb{N}$) and applying a pseudo-polynomial time algorithm for mean-payoff games (e.g., [6]). Intuitively, the trick is that the bounds give strong constraints on the energy levels that can be visited along a play without losing and thus one can restrict the game to a particular graph where acceptable energy levels are encoded in the states and exceeding the bounds is explicitely represented by moving to "losing" states, just as we did in Sect. 3 for the lower bound. Carefully inspecting the construction of [5], we observe that the same construction can be generalized straightforwardly to the multi-dimensional setting. However, the overall complexity is higher: first, the expanded graph will be of *exponential size in k*, the number of dimensions, while still polynomial in S and U. Second, multi-dimensional *limsup* mean-payoff games are in $\mathsf{NP} \cap \mathsf{coNP}$ [25].

Theorem 17. *The threshold problem for multi-dimensional average-energy games with lower and upper bounds is in* $\mathsf{NEXPTIME} \cap \mathsf{coNEXPTIME}$. *That is, given a finite k-dimensional game $G = (S_0, S_1, E)$, an initial state $s_{\mathsf{init}} \in S$, an upper bound $U \in \mathbb{N}^k$, and a threshold $t \in \mathbb{Q}^k$, determining if P_0 has a winning strategy from s_{init} for objective $Energy_{LU}(U) \cap AvgEnergy(t)$ is in* $\mathsf{NEXPTIME} \cap \mathsf{coNEXPTIME}$.

Whether the EXPTIME-hardness that trivially follows from the one-dimensional case [5] can be enhanced to meet this upper bound (or conversely) is an open problem.

Average-energy games with lower bound but no upper bound. Finally, we consider the core setting of this paper, which we just proved decidable in one-dimension, solving the open problem of [5]. Unfortunately, we show that those games are undecidable *as soon as two-dimensional weights are allowed*. To prove it, we reuse some ideas of the proof of undecidability for multi-dimensional total-payoff games presented in [10], but specific gadgets need to be adapted.

Theorem 18. *The threshold problem for lower-bounded average-energy games with two or more dimensions is undecidable. That is, given a finite k-dimensional game $G = (S_0, S_1, E)$, for $k \geq 2$, an initial state $s_{\mathsf{init}} \in S$, and a threshold $t \in \mathbb{Q}^k$, determining whether P_0 has a winning strategy from s_{init} for objective $\mathit{AvgEnergy}_L(t)$ is undecidable.*

References

1. Bloem, R., Chatterjee, K., Henzinger, T.A., Jobstmann, B.: Better quality in synthesis through quantitative objectives. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 140–156. Springer, Heidelberg (2009). doi:10.1007/978-3-642-02658-4_14
2. Boros, E., Elbassioni, K., Gurvich, V., Makino, K.: Markov decision processes and stochastic games with total effective payoff. In: Mayr , E.W., Ollinger, N. (eds.) STACS 2015, LIPics, vol. 30, pp. 103–115. Leibniz-Zentrum für Informatik (2015)
3. Bouyer, P., Fahrenberg, U., Larsen, K.G., Markey, N., Srba, J.: Infinite runs in weighted timed automata with energy constraints. In: Cassez, F., Jard, C. (eds.) FORMATS 2008. LNCS, vol. 5215, pp. 33–47. Springer, Heidelberg (2008). doi:10.1007/978-3-540-85778-5_4
4. Bouyer, P., Hofman, P., Markey, N., Randour, M., Zimmermann, M.: Bounding average-energy games. arXiv:1610.07858 (2016)
5. Bouyer, P., Markey, N., Randour, M., Larsen, K.G., Laursen, S.: Average-energy games. Acta Informatica (2016, in press)
6. Brim, L., Chaloupka, J., Doyen, L., Gentilini, R., Raskin, J.-F.: Faster algorithms for mean-payoff games. Formal Methods Syst. Des. **38**(2), 97–118 (2011)
7. Cassez, F., Jessen, J.J., Larsen, K.G., Raskin, J.-F., Reynier, P.-A.: Automatic synthesis of robust and optimal controllers – an industrial case study. In: Majumdar, R., Tabuada, P. (eds.) HSCC 2009. LNCS, vol. 5469, pp. 90–104. Springer, Heidelberg (2009). doi:10.1007/978-3-642-00602-9_7
8. Chakrabarti, A., Alfaro, L., Henzinger, T.A., Stoelinga, M.: Resource interfaces. In: Alur, R., Lee, I. (eds.) EMSOFT 2003. LNCS, vol. 2855, pp. 117–133. Springer, Heidelberg (2003). doi:10.1007/978-3-540-45212-6_9
9. Chatterjee, K., Doyen, L.: Energy parity games. Theor. Comput. Sci. **458**, 49–60 (2012)
10. Chatterjee, K., Doyen, L., Randour, M., Raskin, J.-F.: Looking at mean-payoff and total-payoff through windows. Inf. Comput. **242**, 25–52 (2015)

11. Chatterjee, K., Randour, M., Raskin, J.-F.: Strategy synthesis for multi-dimensional quantitative objectives. Acta Informatica **51**(3–4), 129–163 (2014)
12. Chatterjee, K., Velner, Y.: Mean-payoff pushdown games. In: LICS 2012, pp. 195–204. IEEE Computer Society (2012)
13. Ehrenfeucht, A., Mycielski, J.: Positional strategies for mean payoff games. Int. J. Game Theor. **8**(2), 109–113 (1979)
14. Fearnley, J., Jurdzinski, M.: Reachability in two-clock timed automata is PSPACE-complete. Inf. Comput. **243**, 26–36 (2015)
15. Fridman, W., Zimmermann, M.: Playing pushdown parity games in a hurry. In: Faella, M., Murano, A. (eds.) GandALF 2012, EPTCS, vol. 96, pp. 183–196 (2012)
16. Hunter, P.: Reachability in succinct one-counter games. arXiv:1407.1996 (2014)
17. Hunter, P.: Reachability in succinct one-counter games. In: Bojańczyk, M., Lasota, S., Potapov, I. (eds.) RP 2015. LNCS, vol. 9328, pp. 37–49. Springer, Cham (2015). doi:10.1007/978-3-319-24537-9_5
18. Juhl, L., Larsen, K.G., Raskin, J.-F.: Optimal bounds for multiweighted and parametrised energy games. In: Liu, Z., Woodcock, J., Zhu, H. (eds.) Theories of Programming and Formal Methods. LNCS, vol. 8051, pp. 244–255. Springer, Heidelberg (2013)
19. Jurdziński, M., Lazić, R., Schmitz, S.: Fixed-dimensional energy games are in pseudo-polynomial time. In: Halldórsson, M.M., Iwama, K., Kobayashi, N., Speckmann, B. (eds.) ICALP 2015. LNCS, vol. 9135, pp. 260–272. Springer, Heidelberg (2015). doi:10.1007/978-3-662-47666-6_21
20. Karp, R.M.: A characterization of the minimum cycle mean in a digraph. Discrete Math. **23**(3), 309–3011 (1978)
21. Larsen, K.G., Laursen, S., Zimmermann, M.: Limit your consumption! Finding bounds in average-energy games. In: Tribastone, M., Wiklicky, B. (eds.), Proceedings 14th International Workshop Quantitative Aspects of Programming Languages and Systems, QAPL 2016, EPTCS, Eindhoven, The Netherlands, vol. 227, pp. 1–14, 2–3 April 2016
22. Niskanen, R., Potapov, I., Reichert, J.: Undecidability of two-dimensional robot games. In: Faliszewski, P., Muscholl, A., Niedermeier, R. (eds.) MFCS 2016, LIPIcs, vol. 58, pp. 73:1–73:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2016)
23. Randour, M.: Automated synthesis of reliable and efficient systems through game theory: a case study. In: Gilbert, T., Kirkilionis, M., Nicolis, G. (eds.) ECCS 2012, Springer Proceedings in Complexity, pp. 731–738. Springer, Heidelberg (2013)
24. Thuijsman, F., Vrieze, O.J.: The bad match; a total reward stochastic game. OR Spektrum **9**(2), 93–99 (1987)
25. Velner, Y., Chatterjee, K., Doyen, L., Henzinger, T.A., Rabinovich, A., Raskin, J.-F.: The complexity of multi-mean-payoff and multi-energy games. Inf. Comput. **241**, 177–196 (2015)
26. Walukiewicz, I.: Pushdown processes: games and model-checking. Inf. Comput. **164**(2), 234–263 (2001)
27. Zwick, U., Paterson, M.: The complexity of mean payoff games on graphs. Theor. Comput. Sci. **158**(1–2), 343–359 (1996)

Logics of Repeating Values on Data Trees and Branching Counter Systems

Sergio Abriola[1,2(✉)], Diego Figueira[3], and Santiago Figueira[1,2]

[1] University of Buenos Aires, Buenos Aires, Argentina
sabriola@dc.uba.ar
[2] ICC-CONICET, Buenos Aires, Argentina
[3] CNRS, LaBRI, Talence, France

Abstract. We study connections between the satisfiability problem for logics on data trees and Branching Vector Addition Systems (BVAS). We consider a natural temporal logic of "repeating values" (LRV) featuring an operator which tests whether a data value in the current node is repeated in some descendant node.

On the one hand, we show that the satisfiability of a restricted version of LRV on ranked data trees can be reduced to the coverability problem for Branching Vector Addition Systems. This immediately gives elementary upper bounds for its satisfiability problem, showing that restricted LRV behaves much better than downward-XPath, which has a non-primitive-recursive satisfiability problem.

On the other hand, satisfiability for LRV is shown to be reducible to the coverability for a novel branching model we introduce here, called Merging VASS (MVASS). MVASS is an extension of Branching Vector Addition Systems with States (BVASS) allowing richer merging operations of the vectors. We show that the control-state reachability for MVASS, as well as its bottom-up coverability, are in 3ExpTime.

This work can be seen as a natural continuation of the work initiated by Demri, D'Souza and Gascon for the case of data words, this time considering *branching* structures and counter systems, although, as we show, in the case of data trees more powerful models are needed to encode satisfiability.

1 Introduction

Logics for data trees. Finite data trees are ubiquitous structures that have attracted much attention lately. A *data tree* is a finite tree whose every position carries a *label* from a finite alphabet and a collection of *data values* from some infinite domain.[1] This structure has been considered in the realms of semi-structured data as a simple abstraction of XML documents, timed automata,

We thank STIC AmSud, ANPCyT-PICT-2013-2011, UBACyT 20020150100002BA, and the Laboratoire International Associé "INFINIS".

[1] Other works have considered different simplifications of these structures, either having only one data value per node (*e.g.*, [2]) or ignoring the label (*e.g.*, [7]).

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 196–212, 2017.
DOI: 10.1007/978-3-662-54458-7_12

program verification, and generally in systems manipulating data values. Finding decidable logics or automata models over data trees is a fundamental quest when reasoning on data-driven systems.

A wealth of specification formalisms on these structures (either for data trees or its 'word' version, data words) have been introduced, stemming from automata [25,28], first-order logic [2,4,16,19], XPath [13–15,18,20], or temporal logics [7,9,11,21,22,24]. In full generality, most formalisms lead to undecidable reasoning problems and a well-known research trend consists of finding a good trade-off between expressiveness and decidability.

Interesting and surprising results have been exhibited about relationships between *logics* for data trees and *counter automata* [18–20]. This is why logics for data trees are not only interesting for their own sake but also for their deep relationships with counter systems.

This work. The aim of this work is to study the basic mechanism of "data repetition", common to many logics studied on data trees. For this, we study a basic logic that can navigate the structure of the tree through the use of CTL-like modalities, and on the other hand can make "data tests", by asking whether a data value is repeated in the subtree. More concretely, the data tests are formulas of the form $u \approx \mathrm{EF}v$, stating that the data value stored in attribute (also called *variable* here) u of the current node is equal to the data value stored in attribute v of some descendant. This *logic of repeating values*, or LRV, has been the center of a line of investigation studied in [6,7] on *data words*, evidencing tight correspondences between reachability problems for Vector Addition Systems and the satisfiability problem. The current work pursues this question further, exhibiting connections between the satisfiability problem of LRV *over data trees* and the bottom-up coverability problem for branching counter systems. In order to obtain connections with branching Vector Addition Systems with States, or branching VASS [29], we also introduce a restriction where tests of the form $u \approx \mathrm{EF}v$ are only allowed when $u = v$. We denote this restriction by $\mathrm{LRV}^{\mathrm{D}}$. This symbiotic relation between counter systems and logics leads us to consider some natural extensions of both the logic and the branching counter systems. In particular, we introduce a new model of branching counter system of independent interest, with decidable coverability and control-state reachability problems, that captures LRV.

The extension of the logic LRV from words to trees is a very natural one. However, the techniques needed to encode the satisfiability of the logic into a counter system are not simple extensions from the ones provided on data words. The reason for this difficulty is manyfold: (*a*) the fact that now the future is non-linear in addition to the possibility of having a data value repeating at several descendants in *different* variables, makes the techniques of [7] for propagating values of configurations impractical; (*b*) further, this seems to be impossible for the case of data trees, and we could only show a reduction for the fragment $\mathrm{LRV}^{\mathrm{D}}$; (*c*) in order to reduce the satisfiability problem for the full logic we will need to augment the power of branching VASS with the possibility to 'merge'

counters in a more powerful way, somewhat akin to what has been done for encoding the satisfiability for FO^2 [19].

Contributions. The main contributions are the following:

- We show that the satisfiability for $\mathrm{LRV}^{\mathrm{D}}$ on k-ranked data trees is reducible, in exponential space, to the control-state reachability problem for VASS_k (*i.e.*, Branching VASS of rank k) in Sect. 5. Since the control-state reachability problem is decidable [29] in 2ExpTime [8], this reduction yields a decision procedure.
- We consider the addition of an operator $\mathrm{AG}_{\approx v}(\varphi)$ expressing "every descendant with the same v-attribute verifies φ", and we show that the logic resulting from adding positive instances of this operator is equivalent to the control-state reachability for Branching VASS, that is, there are reductions in both directions (Sect. 6).
- We introduce an extension of Branching VASS, called *Merging* VASS or MVASS in Sect. 4.2. This model allows for merging counters in branching rules in a form which is not necessarily component-wise, allowing for some weak form of counter transfers. We show that the bottom-up coverability (and control-state reachability) problem for MVASS is in 3ExpTime (Sect. 4.4). This is arguably a model of independent interest.
- We show that the satisfiability for LRV on k-ranked data trees can be reduced to the control-state reachability for MVASS_k in Sect. 7. As in the case of $\mathrm{LRV}^{\mathrm{D}}$, this yields a decision procedure.

Related work. The most closely related work is the one originated by Demri et al. in [5,6] and pursued in [7]. These works study the satisfiability problem for temporal logics on *data words*, extended with the ability to test whether a data value is repeated in the past/future. Indeed, our current work is motivated by the deep correlations evidenced by these works, between counter systems and simple temporal logics on data words. The present manuscript expands this analysis to *branching* logics and counter systems.

There are several works showing links between reachability-like problems for counter systems and the satisfiability problem of logics on data trees. The first prominent example is that satisfiability for Existential MSO with two variables on data words ($\mathrm{EMSO}^2(+1, <, \sim)$) corresponds precisely to reachability of VASS [3], in the sense that there are reductions in both directions. On the other hand, EMSO^2 over (unranked) *data trees* was shown to have tight connections with the reachability problem for an *extension* of BVASS [19], called 'EBVASS'. This extension has features which are very close to the model we introduce here, MVASS, but it does not capture, nor is captured by, MVASS. One can draw a parallel between the situation of the satisfiablity for EMSO^2 and for LRV: while on data words both are inter-reducible to VASS, the extension to data-trees is non-trivial, and they no longer correspond to BVASS, but to *extensions* thereof.

In the course of the last decade, several logics for data trees have been proposed. Among those that feature navigation in terms of modalities such as

temporal operators, one noticeable logic is that of XPath. Although the satisfiability problem for XPath is undecidable, several fragments have been shown to be decidable through reduction to the reachability or coverability problems for counter systems [9,12,18]. In particular, the satisfiability problem for XPath with strict descendant (usually written $\downarrow_+$) on ranked data trees has already a non-primitive-recursive lower bound in complexity, as can be seen by adapting techniques shown for data words [17].

Modulo a simple coding, our logic LRV is captured by a fragment of regular-XPath, here called reg-XPath$_{\mathrm{LRV}}$, on data trees where path expressions are allowed to use Kleene star on any expression (this what we denote by 'regular' XPath), and data tests are of the form $\langle \varepsilon \star \downarrow^* [\varphi] \rangle$ or $\langle \downarrow^n [\varphi] \star \downarrow^m [\psi] \rangle$ for some $n, m \in \mathbb{N}$ and $\star \in \{=, \neq\}$. There are, however, three provisos for this statement. First, in the aforementioned works on XPath the data model consists of data trees whose every position carries exactly *one* data value. In the present paper, we study 'multi-attributed' data trees where, essentially, each node carries a set of pairs 'attribute:value'. However, by means of a simple coding, such as putting every 'attribute:value' as a leaf of the corresponding node, one can easily translate LRV formulas to XPath formulas. Second, our LRV formulas are of the form $u \approx \mathrm{EF} v$ stating that the current data value under attribute u is repeated in a node x of the subtree under attribute v, but one cannot test that some property ψ further holds at the repeating node x. However, it was shown in [7] that one can extend the logic with this power, obtaining formulas of the form $x \approx \mathrm{EF} y[\psi]$, since this extended logic is PTime-reducible to the logic without these tests. Third, the LRV formulas cannot test for regular properties on the labeling of paths, and thus there is no precise characterization in terms of a natural fragment of regular-XPath, but one could add regular path tests to LRV to match the expressive power of reg-XPath$_{\mathrm{LRV}}$ without changing any of our results.

In fact, the fragment reg-XPath$_{\mathrm{LRV}}$ extends also the fragment DataGL considered in [1,13] containing only data tests of the form $\langle \varepsilon \star \downarrow^* [\varphi] \rangle$, which is known to be PSpace-complete on unranked data trees [13].

It is not hard to see that the satisfiability problem of LRV on unranked data trees is PSpace-complete following the techniques from [13]. On the other hand, on ranked data trees we know, by the discussion above, that if we would allow intermediate tests in a way to be able to encode the expressive power of XPath($\downarrow_+$) we would have a non-primitive recursive lower bound. It is therefore natural to limit the navigation *disallowing* intermediary tests. This natural fragment was already studied on data words [7], and we now study it on data trees.

2 Preliminaries

Let $\mathbb{N}^+ = \{1, 2, \dots\}$, $\mathbb{N} = \mathbb{N}^+ \cup \{0\}$, and $\underline{n} = \{1, \dots, n\}$ for every $n \in \mathbb{N}$. We use the bar notation $\bar{x}$ to denote a tuple of elements, where $\bar{x}[i]$, for $i > 0$, refers to the i-th element of the tuple. For any pair of vectors $\bar{x}, \bar{y} \in \mathbb{Z}^k$ we write $\bar{x} \leq \bar{y}$

if $\bar{x}[i] \leq \bar{y}[i]$ for all $1 \leq i \leq k$. The constant $\bar{\emptyset}$ refers to the (unique) vector of dimension 0, and the constant $\bar{e}_i$ refers to the vector (whose dimension will always be clear from the context) so that $\bar{e}_i[i] = 1$ and $\bar{e}_i[j] = 0$ for all $j \neq i$. We write $\bar{0}$ for the tuple of all 0's (the dimension being implicit from the context).

A **linear set** of dimension k is a subset of $\mathbb{N}^k$ which is either empty or described as $\{\bar{v}_0 + \alpha_1\bar{v}_1 + \cdots + \alpha_n\bar{v}_n \mid \alpha_1, \ldots, \alpha_n \in \mathbb{N}\}$ for some $n \in \mathbb{N}$ and $\bar{v}_0, \ldots, \bar{v}_n \in \mathbb{N}^k$. Henceforward we assume that linear sets are represented by the **offset** $\bar{v}_0$ and the **generators** $\bar{v}_1, \ldots, \bar{v}_n$, where numbers are represented in binary. For ease of writing we will denote a linear set like the one above by "$\bar{v}_0 + \{\bar{v}_1, \ldots, \bar{v}_n\}^*$".

We fix once and for all an infinite domain of **data values** $\mathbb{D}$. A **data tree** of rank k over a finite set of labels $\mathbb{A}$ and a finite set of attributes $\mathbb{V}$, is a finite tree whose every node x contains a pair $(a, \mu) \in \mathbb{A} \times \mathbb{D}^{\mathbb{V}}$ and has no more than k children. In general, a will be called the **label** of x and $\mu(v)$ will be called the **data value** of attribute $v \in \mathbb{V}$ at x. The i-**ancestor** of a node x of a data tree T is the ancestor at distance i from x (*i.e.*, the 1-ancestor is the parent); while the i-**descendant** of x are all the descendants of x at distance i.

3 Logic of Repeating Values on Data Trees

We will work with a temporal logic using CTL* modalities [10,26] to navigate the tree—although this is not really essential to our results, in the sense that any other MSO definable data-blind operators could also be added to the logic obtaining similar results. The **Logic of Repeating Values** LRV contains the typical modalities from CTL*, such as EF, AF, EU, etc. as well as the possibility to test for the label of the current node, and *data tests*. Data tests are restricted to being very basic, as in [6], of the form "$u \approx \mathrm{EF}v$" stating "the data value of attribute u appears again at the attribute v of some descendant", or "$u \not\approx \mathrm{EF}v$" stating "there is a descendant node whose attribute v contains a different data value from the data value of the attribute u of the current node". Since LRV is closed under Boolean connectives, this means we can also express, for instance, that attribute u of all descendants have the same data value as the current node's: $\neg(u \not\approx \mathrm{EF}u)$.

Formally, formulas of LRV are defined by

$$\varphi ::= a \mid \varphi \wedge \varphi \mid \neg\varphi \mid \mathrm{EU}(\varphi, \psi) \mid u \approx \mathrm{EF}v \mid u \not\approx \mathrm{EF}v \mid u \approx \mathrm{EX}^i v \mid u \not\approx \mathrm{EX}^i v,$$

where a ranges over a finite set of labels $\mathbb{A}$, u, v range over a finite set of attribute variables $\mathbb{V}$ (also called just 'variables'), and $i \in \mathbb{N}^+$. Given a data tree T and a node x of T, the satisfaction relation $\models$ is defined in the usual way: $T, x \models a$ if a is the label of x; $T, x \models u \approx \mathrm{EF}v$ [resp. $T, x \models u \not\approx \mathrm{EF}v$] if there is a strict descendant y of x so that the u-attribute of x has the same [resp. different] data value as the v-attribute of y; $T, x \models u \approx \mathrm{EX}^i v$ [resp. $T, x \models u \not\approx \mathrm{EX}^i v$] if there exists an i-descendant of x whose v-attribute is equal [resp. distinct] to the u-attribute of x; and $T, x \models \mathrm{EU}(\varphi, \psi)$ if there is some strict descendant y of x so that $T, y \models \varphi$

and every other node z strictly between x and y verifies $T, z \models \psi$. Note that the remaining CTL* modalities (EX, EG, EF, AX, AG, AF, AU) can be expressed using EU[2].

We call $\mathrm{LRV}^{\mathrm{D}}_n$ the logic using at most n attribute variables, whose only admissible data tests are of the form $u \approx \mathrm{EF}u$, $u \not\approx \mathrm{EF}u$, $u \approx \mathrm{EX}^i u$ or $u \not\approx \mathrm{EX}^i u$ (same variable in the left and right sides). Intuitively, this corresponds to the restriction where each attribute variable ranges over a *disjoint* set of data values (hence the letter 'D').

4 Models of Branching Counter Systems

We present the models of counter systems we are going to work with. The first one is a well-known model, usually known as Branching Vector Addition System with States, or "BVASS", while the second one is a useful extension of the first one where the split/merge operation of the counters is controlled by the use of linear sets.

4.1 Branching VASS

A VASS of rank k and dimension n, or $n\mathrm{VASS}_k$, is a tuple $\mathcal{A} = \langle Q, U, B\rangle$, where Q is a finite set of states, $U \subseteq Q \times \mathbb{Z}^n \times Q$ is a set of unary rules, and $B \subseteq Q \times Q^{\leq k}$ is a finite set of branching rules. We notate $q \xrightarrow{\bar{v}} q'$ for a unary rule $(q, \bar{v}, q') \in U$, and $q \to (q_1, \ldots, q_i)$ for a branching rule $(q, q_1, \ldots, q_i) \in B$. A **configuration** is an element from $\mathit{Confs} := Q \times \mathbb{N}^n$. For a configuration $(q, \bar{n})$ we often use the term "counter i" instead of "$n[i]$" (in the case $n = 1$ we speak of *the* counter).

A **derivation tree** [resp. **incrementing derivation tree**] is a finite tree $\mathcal{D}$ whose every node x is either

- labeled with a pair $(p \xrightarrow{\bar{v}} p', (q, \bar{n})) \in U \times \mathit{Confs}$ so that $p \xrightarrow{\bar{v}} p'$ is a unary rule of U, $p = q$ and it has exactly one child, which is labeled $(r_1, (p_1, \bar{n}_1))$ so that $p' = p_1$ and

$$\bar{n} + \bar{v} = \bar{n}_1 \quad [\text{resp. } \bar{n} + \bar{v} \leq \bar{n}_1]; \tag{1}$$

- or labeled with a pair $((p, \bar{q}), (q, \bar{n})) \in B \times \mathit{Confs}$ so that $p \to \bar{q}$, with $\bar{q} \in Q^{k'}$ for some $k' \leq k$, is a branching rule of B, $p = q$ and it has exactly k' children, labeled $(r_1, (p_1, \bar{n}_1)), \ldots, (r_{k'}, (p_{k'}, \bar{n}_{k'}))$ so that $\bar{q} = (p_1, \ldots, p_{k'})$ and

$$\bar{n} = \sum_{i \leq k'} \bar{n}_i \quad [\text{resp. } \bar{n} \leq \sum_{i \leq k'} \bar{n}_i]. \tag{2}$$

Note that leaf nodes are necessarily labeled with rules of the form $q \to \bar{\emptyset} \in B$. Without loss of generality we will assume that the system contains rules $q \to \bar{\emptyset}$ for every state q.

[2] $\mathrm{EX}\varphi = \mathrm{EU}(\varphi, \bot)$, $\mathrm{EF}\varphi = \mathrm{EU}(\varphi, \top)$, $\mathrm{EG}\varphi = \mathrm{EU}(\varphi \wedge \neg\mathrm{EX}\top, \varphi)$, $\mathrm{AU}(\varphi, \psi) = \neg\mathrm{EU}(\neg\psi \wedge \neg\varphi, \neg\psi) \wedge \neg\mathrm{EG}(\neg\varphi)$, etc.

4.2 Merging VASS

We present an extension of the model above where the branching rules, now called *merging rules*, are more powerful: they allow us to reorganize the counters. Whereas in an (incrementing) derivation tree for VASS_k the component i of the configuration of a node depends only on the component i of its children and the rule applied, MVASS_k allows to have transfers *between components*. However, these transfers have some restrictions—otherwise the model would have non-elementary or undecidable coverability/reachability problems [23]. First, transfers between components are 'weak', in the sense that we cannot force a transfer of the whole value of a coordinate i to a distinct coordinate j of a child, we can only make sure that part of it will be transferred to component j and part of it will remain in component i. Second, these weak transfers can only be performed for any pair of coordinates i, j adhering to a partial order, where transfers occur from a bigger component to a smaller one.

A **Merging-VASS** of rank k and dimension n, or $n\mathrm{MVASS}_k$, is a tuple $\mathcal{A} = \langle Q, U, M, \preceq \rangle$, where $\preceq$ is partial order on $\underline{n}$, Q and U are as before, and M is a set of merging rules of the form $(q, S, \bar{q})$ where $q \in Q$, $\bar{q} \in Q^{k'}$ with $k' \leq k$, and $S \subseteq \mathbb{N}^{n\cdot(k'+1)}$ is a linear set of dimension $n\cdot(k'+1)$ of the form $\bar{0} + (B \cup S_0)^*$, where

1. all the elements of B are of the form $(\bar{e}_i, \bar{x}_1, \ldots, \bar{x}_{k'})$, where for each $1 \leq \ell \leq k'$, $\bar{x}_\ell \in \mathbb{N}^n$ is either $\bar{0}$ or $\bar{e}_j$ for some $j \prec i$; and
2. S_0 consists of the following $k' \cdot n$ vectors

$$S_0 = \bigcup_{1 \leq i \leq n} \{(\bar{e}_i, \bar{e}_i, \bar{0}, \bar{0}, \ldots, \bar{0}), (\bar{e}_i, \bar{0}, \bar{e}_i, \bar{0}, \ldots, \bar{0}), \ldots, (\bar{e}_i, \bar{0}, \ldots, \bar{0}, \bar{e}_i)\}. \quad (3)$$

The idea is that in point 1 we allow to transfer contents from component i to components of smaller order. For example, on dimension 3 and rank 2, a vector $\bar{v} = (1,0,0)(0,1,0)(0,0,1)$ in B would imply that during the merge one can transfer a quantity $m > 0$ from component 1 of the father into component 2 of the first child and component 3 of the second child, assuming $2, 3 \prec 1$). On the other hand, point 2 tells us that for every i we can always have some quantity of component i that is *not* transferred to other components, *i.e.*, that stays in component i. Continuing our example, the children configurations $(m, m' + s, t)$ and $(m, s, m' + t)$ can be merged into $(m + m', s, t)$ for every $m, m', s, t \geq 0$, using the vector $\bar{v}$ and S_0.

A derivation tree [resp. incrementing derivation tree] is defined just as before, with the sole difference being that condition (2) is replaced with

$$\begin{aligned} &(\bar{n}, \bar{n}_1, \ldots, \bar{n}_{k'}) \in S \\ &[\text{resp. } (\bar{n}, \bar{n}'_1, \ldots, \bar{n}'_{k'}) \in S \text{ for } (\bar{n}'_1, \ldots, \bar{n}'_{k'}) \leq (\bar{n}_1, \ldots, \bar{n}_{k'})]. \end{aligned} \quad (4)$$

Notice that this is a generalization of VASS_k. Indeed, VASS_k corresponds to the restriction where all the k'-ary merging rules have $S = \bar{0} + S_0^*$ for S_0 as defined in (3). Note that an (incrementing) derivation tree for $n\mathrm{VASS}_k$ is, in particular, an

(incrementing) derivation tree for nMVASS$_k$. As before, we assume that there are always rules $(q, \emptyset, \bar{\emptyset})$ for every state q.

Jacquemard et al. [19] study an extension of BVASS, 'EBVASS', in relation to the satisfiability of FO$^2(<, +1, \sim)$ over *unranked* data trees. EBVASS has some features for merging counters. While MVASS and EBVASS are incomparable in computational power, it can be seen that without the restriction $j \prec i$ in condition 1, MVASS would capture EBVASS. In fact, this condition is necessary for the (elementary) decidability of the coverability problem for MVASS, while the status of the coverability problem for EBVASS is unknown.

4.3 Decision Problems

Given a counter system $\mathcal{A}$, a set of states $\widehat{Q}$, and a configuration $(q, \bar{n})$ of $\mathcal{A}$, we write $(q, \bar{n}) \rightsquigarrow_{\mathcal{A}} \widehat{Q}$[resp. $(q, \bar{n}) \rightsquigarrow^+_{\mathcal{A}} \widehat{Q}$] if there exists a derivation tree [resp. incrementing derivation tree] for $\mathcal{A}$ with root configuration $(q, \bar{n})$, so that all the leaves have configurations from $\widehat{Q} \times \{\bar{0}\}$. The reachability and incrementing reachability problems are defined as follows.

Problem:	VASS$_k$ **reachability problem** [resp. VASS$_k$ **incrementing reachability problem**]
Input:	an nVASS$_k$ $\mathcal{A}$ with states Q, a set of states $\widehat{Q} \subseteq Q$, and a configuration $(q, \bar{n})$ of $\mathcal{A}$
Output:	'Yes' iff $(q, \bar{n}) \rightsquigarrow_{\mathcal{A}} \widehat{Q}$ [resp. $(q, \bar{n}) \rightsquigarrow^+_{\mathcal{A}} \widehat{Q}$]

Observe that when $k = 1$ this problem is equivalent to the reachability and coverability problems for Vector Addition Systems with States.

The **MVASS$_k$ reachability problem** and **MVASS$_k$ incrementing reachability problem** are defined just as before but considering $\mathcal{A}$ to be an nMVASS$_k$ instead of a nVASS$_k$. We will often refer to these problems as Reach( ) and Reach$^+$( ). We also remark that the incrementing reachability problem is simply a restatement of the *coverability problem*. In particular, it is monotone: if $(q, \bar{n}) \rightsquigarrow^+_{\mathcal{A}} \widehat{Q}$ and $\bar{n}' \leq \bar{n}$ then $(q, \bar{n}') \rightsquigarrow^+_{\mathcal{A}} \widehat{Q}$. We define the **control-state reachability problem** CSReach as the problem of, given $\mathcal{A}, q, \widehat{Q}$, whether $(q, \bar{n}) \rightsquigarrow_{\mathcal{A}} \widehat{Q}$ for some $\bar{n}$. It is easy to see that this problem is equivalent to the problem of whether $(q, \bar{0}) \rightsquigarrow^+_{\mathcal{A}} \widehat{Q}$.

In [8] the coverability problem (or equivalently, the incrementing reachability problem) for a single-state formulation, called BVAS, is studied. A BVAS consists of a tuple $\langle n, R_1, R_2 \rangle$, where R_1 is a set of unary rules, R_2 is a set of binary rules (both rules included in $\mathbb{Z}^n$ which add up a vector). The *size* of a given BVAS is defined as $n\ell$, where ℓ represents the maximum binary size of an entry in $R_1 \cup R_2$.

Proposition 1. *[8] Coverability for* BVAS *is 2ExpTime-complete. If the dimension n is fixed, the problem is in ExpTime.*

4.4 Decidability of Reach$^+$(MVASS)

The arguments used in [8] to prove the previous proposition can be adapted to show a similar result for MVASS: the Reach$^+$ and CSReach problems are in 3ExpTime.

Theorem 2. *Reach$^+$(MVASS$_k$) and CSReach(MVASS$_k$) are in 3ExpTime for every $k \geq 1$. If the dimension n is fixed, the problem is in 2ExpTime.*

Proof (idea). In a somewhat similar way as it was shown in [8, Lemma 6] for the case of VASS$_k$, one can show that if there is an incrementing derivation $\mathcal{D}$ witnessing $(q, \bar{n}) \rightsquigarrow^+_{\mathcal{A}} \widehat{Q}$, where $\mathcal{A} = (Q, U, M, \preceq)$ is an nMVASS$_k$ counter system, then there is a 'contraction' (*i.e.*, the result of the repeated replacing of the subtree at a node x with the subtree at some descendant y while maintaining the property of being a derivation) $\mathcal{D}'$ of $\mathcal{D}$ with height bounded doubly-exponentially in the dimension. One significant difficulty in adapting the proof for VASS$_k$ to MVASS$_k$, is that in a derivation for a VASS$_k$, if the component i of a configuration at a node x is "very big", and the same component i at the root is "small" (say, 0), this means that the distance between x and the root in the derivation tree must be big. This is a crucial ingredient for bounding the height of a minimal derivation and obtaining the 2ExpTime upper bound proof for Reach$^+$(VASS$_k$) in [8]. However, this is no longer the case for MVASS$_k$, since the size of component i at x may come from a transfer of the parent from another component with higher $\preceq$-index. Nevertheless, by using the fact that (i) transfers between components induced by merging rules are $\preceq$-ordered (point 1 of the definition), and (ii) linear sets of merging rules are monotone in the sense that they contain S^*_0 as defined in (3), we can recover bounds for the minimal height of derivations. This can be done by induction on the preorder—note that relative to maximal $\preceq$ coordinates MVASS$_k$ behaves like VASS$_k$.

These results imply that, in order to decide the incrementing reachability problem, it suffices to search for a derivation of doubly-exponential height, whose vectors may contain triply-exponential entries in principle. As a consequence of this, the verification of the existence of such a derivation can be performed in alternating double exponential space, as it is shown in [8, Theorem 8], and thus the incrementing reachability for MVASS is in 3ExpTime.

If n is fixed, the height of the witnessing derivation becomes singly exponential and thus the problem is in 2ExpTime (as explained in [8, Theorem 8]). □

5 Satisfiability of LRV$^{\mathrm{D}}$ on data trees

We call SAT$_k$ the satisfiability problem on finite k-ranked data-trees. The main result of this section is the following.

Theorem 3. *SAT$_k$-LRV$^{\mathrm{D}}_n$ is ExpSpace-reducible to* CSReach(nVASS$_k$).

In the proof of the theorem, the number of attribute variables of the formula will become the dimension of the VASS$_k$. Since the CSReach problem for VASS$_k$

is decidable in 2ExpTime, this yields a decidable procedure for SAT_k-$\mathrm{LRV}^{\mathrm{D}}$ for every k. For the case $k = 1$, *i.e.*, on *data words*, it has been shown [7] that there is a reduction from SAT_1-LRV_n to $\mathrm{CSReach}(2^n\text{-}\mathrm{VASS}_1)$, where the dimension of the VASS_1 is exponential in the number of variables. However, it is easy to see that the proof of [7] also yields a reduction from SAT_1-$\mathrm{LRV}^{\mathrm{D}}_n$ to $\mathrm{CSReach}(n\mathrm{VASS}_1)$. Thus, this theorem has been shown for $k = 1$, and here we generalize it to $k > 1$. However, there are a number of problems that appear if one tries to "extend" the proof of [7] to the branching setup. In particular, the non-linearity of the future in addition to the possibility of having a data value repeating at several descendants in different variables, calls for a non-standard way of propagating the values of configurations, which is not contemplated in VASS_k. This is why we are only able to show the reduction for the 'disjoint' fragment $\mathrm{LRV}^{\mathrm{D}}$, and which leads us to consider the extended model MVASS_k in Sect. 7. This propagation problem does not appear when one only considers that the classes of different values are disjoint, that is, that all formulas of the type $v \star \mathrm{EF} w$ with $\star \in \{\approx, \not\approx\}$ have $v = w$, motivating the study of SAT_k-$\mathrm{LRV}^{\mathrm{D}}_n$.

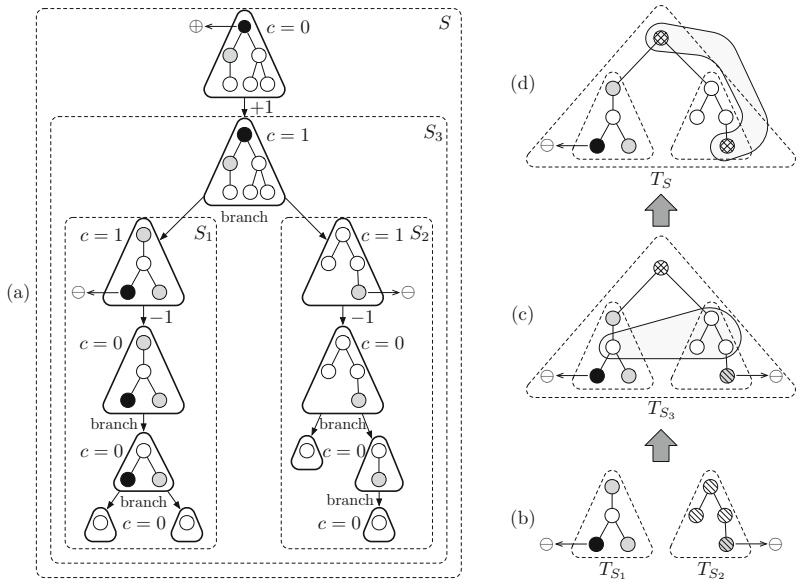
Proof idea. We start by analyzing a restricted case, which serves as building block: the logic $\mathrm{LRV}^{\mathrm{D}-}_1$ whose only formulas are conjuncts of terms of the form $v \star \mathrm{EX}^i v$, $v \star \mathrm{EF} v$, or their negation, where $\star \in \{\approx, \not\approx\}$. We show that for any formula φ of $\mathrm{LRV}^{\mathrm{D}-}_1$, there is a $1\mathrm{VASS}_k$ $\mathcal{A}^k_\varphi = \langle Q, U, B\rangle$, a set of initial states $Q_0 \subseteq Q$, and a set of final states $\widehat{Q} \subseteq Q$ such that $\mathrm{SAT}_k(\varphi)$ iff there is a derivation tree with a starting node in $q_0 \in Q_0$ that is a solution to $\mathrm{CSReach}(\mathcal{A}^k_\varphi, q_0, \widehat{Q})$—it is easy to see that this problem is equivalent to CSReach as stated in Sect. 4.3. We then extend this construction to the automaton $\mathcal{B}^k_\varphi$, enabling a reduction from the full logic $\mathrm{LRV}^{\mathrm{D}}_1$, but still restricted to only one variable. Finally, because of the disjointness of the variables, it is easy to extend these constructions to the full logic $\mathrm{LRV}^{\mathrm{D}}_n$.

Here we only give a brief explanation of the construction of $\mathcal{A}^k_\varphi = \langle Q, U, B\rangle$ for the logic $\mathrm{LRV}^{\mathrm{D}-}_1$. For the sake of simplicity, we assume our logic has no labels; their addition to the construction is straightforward. Since LRV can only deal with data (in)equality and since in this case we consider $n = 1$, we will interchangeably speak of an equivalence relation between the nodes of the tree or of the particular data values.

We define the EX-length of a formula ψ as the maximum i such that ψ contains a subformula of the form $v \star \mathrm{EX}^i v$. Let d be the EX-length of φ. The set Q consists of all valid (d, k)-frames, where a (d, k)-frame is a tree of depth d and rank k, equipped with an equivalence relation, and with some extra attributes (node-labeling functions) to include some special marks and semantic information of future requirements of the form $(\neg)v \approx \mathrm{EF} v$ and $(\neg)v \not\approx \mathrm{EF} v$. The initial states Q_0 are those frames F satisfying the *local* part of φ (that is, subformulas of φ the form $v \star \mathrm{EX}^i v$). *Future requirements* (that is, subformulas of φ of the form $v \star \mathrm{EF} v$) may not be satisfied locally in F. The set $\widehat{Q}$ is the singleton with a frame consisting in a single node. The basic idea is that the counter of $\mathcal{A}^k_\varphi$ keeps track of how many future requirements are not yet satisfied. Some nodes

of the frames may have extra information in the form of labels $\oplus$ or $\ominus$. States F whose root is labeled with $\oplus$ are *points of increment*: $\mathcal{A}^k$ will have unary rule in U that increments the (sole) counter in 1. A point of increment denotes that some subformula $v \approx \mathrm{EF}v$ of φ should hold, but it is not satisfied *locally*, that is, inside F. Leaves with $\ominus$ are those not related to ancestors in the frame with the same data value, they can thus be "joined" into the same equivalence class to a future requirement originated at some distant ancestor. States with leaves $\ominus$-labeled are *points of decrement*: $\mathcal{A}^k$ will have a unary rule in U to decrement the counter depending on the number of equivalence classes of leaves labeled with $\ominus$. The branching rules B of $\mathcal{A}^k$ are of the form $F \to (F_1, \ldots, F_i)$, where F_j overlaps with an adequate part of F.

Example 4. The following figure illustrates a scheme of an incrementing derivation S of the $1\mathrm{VASS}_2$ $\mathcal{A}^2_\varphi$ (a) and some steps (b, c and d) in the bottom up construction of the data tree T_S satisfying φ, for $\varphi = \neg v \approx \mathrm{EX}v \wedge \neg v \approx \mathrm{EX}^2 v \wedge v \approx \mathrm{EF}v$. Triangles represent $(2,2)$-frames. Shades of gray represent the equivalence classes, which only make sense inside any frame. The counter is notated with c, and arrows represent the (unary/branching) transitions of the derivation. Notice that the top branching is 'incremental', and that the local requirements of φ (namely, $\neg v \approx \mathrm{EX}v$ and $\neg v \approx \mathrm{EX}^2 v$) are satisfied in the root of the top frame.



The construction of T_S is bottom-up, and we show three steps: (a), (b) and (c). Notice that in (b) each of T_{S_1} and T_{S_2} has its own partition (no intersection). In (c) we process the root of S_3 by tying together T_{S_1} and T_{S_2} with a common parent, who lives in a single class of the partition. Notice that the partitions of

T_{S_1} and T_{S_2} are properly joined (grey area), according to the information in the root of S_3. Finally in (d) we construct T_S. The root of S is a point of increment, so we match $\oplus$ with some $\ominus$ in T_{S_3}. In this case, we match it with the right-hand $\ominus$, and so we join them by putting them in the same partition (grey area). We have satisfied the future requirement $v \approx \mathrm{EF}v$ of φ.

On the one hand, any incrementing derivation S that is a solution to $\mathrm{CSReach}(\mathcal{A}^k_\varphi, q_0, \widehat{Q})$ for some $q_0 \in Q_0$, can be translated into a data tree T_S whose root satisfies φ. In fact, any semantic information contained in the labels of nodes in frames of S will be satisfied in the corresponding nodes of T_S. The difficult part is to show that $\ominus$-leaves will have the necessary conditions to be joined with the equivalence class of an $\oplus$-ancestor, making true the formula $v \approx \mathrm{EF}v$. This will be a consequence of the fact that the incrementing derivation satisfies CSReach.

On the other hand, if φ is satisfiable in some k-ranked data tree T then we can build an incrementing derivation tree S_T that is a solution to $\mathrm{CSReach}(\mathcal{A}^k_\varphi, q_0, \widehat{Q})$ for some $q_0 \in Q_0$. Following the ideas of the previous part, we proceed from the root toward the leaves using the structure and equivalence classes of T to determine in each step the corresponding states (including the semantic labels and the labels $\ominus$, $\oplus$) and rules of S_T. From the construction, and using the incrementing nature of the derivation, it will follow that S_T is a solution to the control-state reachability problem.

For the construction of $\mathcal{B}^k_\varphi$, the information given by the (d,k)-frames will be supplemented by the addition of sets of formulas containing information about the EU operator and the Boolean connectives. The way to do this is standard (see *e.g.*, [6]).

Complexity. Let $\mathrm{LRV}^{\mathrm{D}}_{n,d}$ be the fragment of $\mathrm{LRV}^{\mathrm{D}}_{n}$ where each formula has EX-length at most d. By inspecting the above reduction, we can bound the number of states of the constructed $n\mathrm{VASS}_k$ by $O(p(n)^{k^{d+1}} \cdot (k^{d+1})^{k^{d+1}} \cdot 2^{p(|\varphi|)})$ for some polynomial p, and we can bound the maximum value among the entries in unary rules by k^d. Furthermore, we can reduce our branching VASS to an equivalent (single-state) BVAS with an addition of a constant number of new dimensions. This transformation increases the binary size of the maximum entry of the unary rules at most logarithmically over the number of states of our original $n\mathrm{VASS}_k$. Now, using Proposition 1, Theorem 3, and the above complexity analysis, we obtain:

Proposition 5. *SAT_k-$\mathrm{LRV}^{\mathrm{D}}_{n,d}$ is in ExpTime for fixed k, n, d; it is in 2ExpTime for fixed k, n or fixed d, k; and it is in 3ExpTime for fixed k.*

6 Obtaining Equivalence with VASS_k

In the previous section we have seen a reduction into the control-state reachability problem for VASS_k. A natural question is whether there exists a reduction in the other direction: can $\mathrm{CSReach}(\mathrm{VASS}_k)$ be reduced into the k-satisfiability

for LRV^{D}? For the case $k = 1$, this has been shown to be the case [7]: there exists a polynomial-space reduction from $\text{CSReach}(\text{VASS}_1)$ to $\text{SAT}_1(\text{LRV})$.

The existence of a reduction would show, intuitively, that one can express in the logic that there is a tree that verifies all the conditions for being a derivation. Without the use of data tests, one can easily encode trees that verify all the conditions except perhaps (1) and (2) regarding the vectors. For this, let us assume without loss of generality that all unary rules contain a vector $\bar{e}_i$ or $-\bar{e}_i$. The data values are used to ensure the next two conditions:

- Along any branch, every node containing a rule of the form $q \xrightarrow{\bar{e}_i} q'$ has a unique data value. In other words, we cannot find two nodes encoding an increment of component i with the same data value so that one is the ancestor of the other.
- For every node with a unary rule $q \xrightarrow{\bar{e}_i} q'$ there exists a descendant with a rule $p \xrightarrow{-\bar{e}_i} p'$ and the same data value.

These two conditions imply that after incrementing component i there must be *at least* one corresponding decrement of component i. Note that there could be *more* decrements than increments, which is not a problem since we work under the 'incrementing' semantics.

Interestingly, these two conditions can be expressed in LRV, but we do not know how to encode it in LRV^{D} (we conjecture that they are not expressible).

Adding the operator $\mathbf{AG}_{\approx v}(\varphi)$. We add a new operator $\text{AG}_{\approx v}(\varphi)$ to LRV^{D}, where $T, x \models \text{AG}_{\approx v}(\varphi)$ if every descendant of x with the same v-attribute verifies φ. The fragment of LRV^{D}_n extended with positive occurrences of $\text{AG}_{\approx v}(\varphi)$ (that is, where $\text{AG}_{\approx}$ occurs always under an even number of negations) is called $\text{LRV}^{\text{D}}_n(\text{AG}^{+}_{\approx})$.

Now, in $\text{LRV}^{\text{D}}_n(\text{AG}^{+}_{\approx})$ one can express: *for every node x containing a rule $q \xrightarrow{\bar{e}_i} q'$, we have that all descendants of x with the same v_i attribute contain a rule of the form $p \xrightarrow{-\bar{e}_i} p'$*. This, added to the property that every increment for component i must verify $v_i \approx \text{EF}v_i$, ensures that the tree indeed encodes a derivation tree.

Theorem 6. $\text{CSReach}(n\text{VASS}_k)$ *is PTime-reducible to* SAT_k-$\text{LRV}^{\text{D}}_1(AG^{+}_{\approx})$.

Proof (idea). We show the idea for $n = 1$, as this case generalizes to any n straightforwardly, and without changing the number of variables in the logic. For every 1VASS_k $\mathcal{C}^k = \langle Q, U, B\rangle$, $q_0 \in Q$ and $\widehat{Q} \subseteq Q$ we define $\varphi \in \text{LRV}^{\text{D}}_1(\text{AG}^{+}_{\approx})$, such that $\text{SAT}_k(\varphi)$ iff $\text{CSReach}(\mathcal{C}^k_\varphi, q_0, \widehat{Q})$. We want this φ to force various properties in all its models, so that every model corresponds to a derivation tree of $\text{CSReach}(\mathcal{C}^k_\varphi, q_0, \widehat{Q})$. In particular, we want:

- Each node is labeled with either a rule of $U \cup B$ or an extra label $*$ for dummy nodes that will be ignored (this is to force exact k-branching for all non-leaves). We can assume without loss of generality that all unary rules in U of the form $q \xrightarrow{c} q'$ have either $c = 1$ or $c = -1$. In particular, formulas φ_{inc} and φ_{dec} express that the label is an increment or a decrement rule, respectively.

- If a node is labeled with an empty rule $q \to \bar{\emptyset}$, then it is a leaf.
- The root is labeled with a rule of the form $(q_0, \dots) \in U \cup B$.
- Each node labeled with an increment rule has a descendant in the same equivalence class (*i.e.* with same value for the only attribute v), and all its descendants in the same equivalence class are labeled with a decrement rule: this can be expressed by $\varphi_{\text{inc}} \to \big(v \approx \text{EF}v \wedge \text{AG}_{\approx v}(\varphi_{\text{dec}})\big)$.

All the above properties, except the last one, can be expressed in LRV^{D}_1; for the last one we use (positively) $\text{AG}_{\approx}$. The final formula φ consists of a conjunction of all these properties, among others (so that the occurrence of $\text{AG}_{\approx}$ remains positive). Then one verifies that a solution to the control-state reachability problem of $\mathcal{C}^k, q_0, \widehat{Q}$ can be used to construct a model for φ; and that a data tree satisfying φ can be used to construct an incrementing derivation tree for $\text{CSReach}(\mathcal{C}^k, q_0, \widehat{Q})$. □

The satisfiability for this extension still has a reduction to the control-state reachability for VASS_k:

Theorem 7. *SAT_k-$\text{LRV}^{\text{D}}(\text{AG}^{+}_{\approx})$ is ExpSpace-reducible to* $\text{CSReach}(\text{VASSR}_k)$.

7 From LRV to MVASS$_k$

The reduction from LRV^{D} to VASS_k from Sect. 5 cannot be extended to treat LRV. The main problem is that the branching nature of the counters in a $\text{CSReach}(\text{VASS}_k)$ will be insufficient to represent some classes of data trees (which can be needed to model some formulas). When we have tests of the form $u_1 \approx \text{EF}u_2$ with $u_1 \neq u_2$ distinct variables, we can no longer reason in terms of "one coordinate i for each variable u_i", where the i-th component in the configuration of the VASS_k codes, intuitively, how many distinct data values must be seen on variable u_i in the subtree as shown in Sect. 5. In fact, when working with LRV, a data value may appear in *several* variables, as a result of allowing formulas like $u_1 \approx \text{EF}u_2 \wedge u_1 \approx \text{EF}u_3$. This means that we need to reason in terms of *sets* of variables, where each component i is associated with a non-empty subset U_i of the variables appearing in the input formula φ; this time, component i counts how many data values must appear in the subtree under all the variables of U_i. This, in principle, poses no problem for the non-branching case: in fact, this kind of coding (indexing one coordinate of the configuration for each subset of variables) was used in [6] to show a reduction from LRV to VASS on data words. However, on data trees, this coding breaks with the semantics of the branching rules of VASS_k.

As an example, suppose we work with two variables u, v and we thus have dimension 3—the first component is associated with $\{u\}$, the second with $\{v\}$ and the third with $\{u, v\}$. Suppose that there are n ancestor nodes that have to satisfy both $u \approx \text{EF}u$ and $u \approx \text{EF}v$, which at the current configuration of the VASS_k is witnessed by the vector $(0, 0, n)$. Intuitively, this means that there are

n data values that must appear in the subtree under a variable u and also under v (though not necessarily at the same node) in the data tree the automaton is trying to find. Hence, as part of the "branching" instruction of this configurations into the configuration of the left and right children, one must contemplate the possibility of obtaining, for instance, $(n, 0, 0)\,(0, n, 0)$, saying that the left subtree contains n distinct data values for u, and the right child contains n data values for v. But it could be $(n - t, 0, t)\,(0, n - t, 0)$, or $(0, 0, n - t)\,(0, 0, t)$, etc. In other words, components need to be mixed in a more complex way that is not allowed in VASS_k branching rules. In particular, some sort of transfers between coordinates must be necessary. This is precisely the behavior that we can encode into MVASS.

Theorem 8. *SAT_k-LRV_n is reducible to* $\text{CSReach}(2^n\text{-MVASSR}_k)$.

Proof (idea). The crux of the reduction is in the use of the *merging* rules. We have one component associated to every non-empty subset. For every non-empty subset U of variables appearing in the input formula φ there is a component i associated to U, let us then define $\bar{e}_U$ as $\bar{e}_i$. Also, let $\bar{e}_\emptyset = \bar{0}$. The idea is that, on rank k, every merging instruction will contain the linear set consisting of every vector $(\bar{e}_U \bar{e}_{V_1} \cdots \bar{e}_{V_{k'}}) \in \mathbb{N}^{n\cdot(k'+1)}$ with $k' \leq k$, so that $U \neq \emptyset$ and $U = \bigcup_i V_i$. The partial order $\preceq$ will then be the subset ordering on the components: $i \preceq j$ if the set associated to i is contained in that associated to j.

Using the merging rules as described above, the reduction from LRV^{D} to VASS_k of Sect. 5 can be modified to obtain a reduction from LRV to MVASS_k. Frames and its notion of validity are extended to treat set of variables. In particular, now the points of increment and decrement are always relative to a set of variables. This follows, very roughly, the idea of coding from [7] in the setup built in Sect. 5, but now some special care must be considered because of the non-linearity of a tree. One must decide in advance to which leaf of the frame the satisfaction of data demands will be delegated. The resulting MVASS_k now has dimension *exponential* in the number of variables of the input formula. □

As a corollary, due to Theorem 2, we have that SAT_k-LRV is decidable. We remark that, similarly as done in [7], one can add formulas of the form $u \star \text{EF}[\varphi]v$ stating that there is a descendant witnessing $u \star \text{EF}v$ and verifying φ, while preserving this reduction.

8 Discussion

We have shown connections between counter systems and data logics on ranked data trees. In particular, this has yielded decision procedures for data logics and a new model of branching computation of VASS.

While in the present work the focus has been put on *ranked* data trees, we envisage working also on unranked trees in the future. In particular, we remark that these logics can be naturally extended to the unranked case, but that there

are no well-known models of branching counter systems with *unbounded* branching. This may lead to new natural models featuring some sort of unbounded parallel computations with good computational properties.

We are also interested in considering other modalities in our logics, with branching tests such as $\mathrm{EX}^i v \star \mathrm{EF} u$ and $\mathrm{EF} u \approx \mathrm{EF} v$, or tests including past such as $u \approx \mathrm{EF}^{-1} v$ and $\mathrm{EF}^{-1} u \approx \mathrm{EF} v$.

We were unable to show the precise complexity of $\mathrm{CSReach}(\mathrm{MVASS}_k)$, which lies between 2ExpTime and 3ExpTime. We leave this for future work. We believe that $\mathrm{SAT}_k\text{-LRV}(\mathrm{AG}^{+}_{\approx})$ is equivalent to the control-state reachability problem for MVASS_k, in the sense of existence of computable reductions from and to.

References

1. Baelde, D., Lunel, S., Schmitz, S.: A sequent calculus for a modal logic on finite data trees. In: 25th EACSL Annual Conference on Computer Science Logic, CSL 29, 1 September 2016, Marseille, France, pp. 32:1–32:16, August 2016
2. Bojańczyk, M., Muscholl, A., Schwentick, T., Segoufin, L.: Two-variable logic on data trees and XML reasoning. JACM **56**(3), 13 (2009)
3. Bojańczyk, M., David, C., Muscholl, A., Schwentick, T., Segoufin, L.: Two-variable logic on data words. ACM Trans. Comput. Log. 12(4) 2010
4. Bollig, B., Cyriac, A., Gastin, P., Narayan Kumar, K.: Model checking languages of data words. In: Birkedal, L. (ed.) FoSSaCS 2012. LNCS, vol. 7213, pp. 391–405. Springer, Heidelberg (2012). doi:10.1007/978-3-642-28729-9_26
5. Demri, S., D'Souza, D., Gascon, R.: A decidable temporal logic of repeating values. In: Artemov, S.N., Nerode, A. (eds.) LFCS 2007. LNCS, vol. 4514, pp. 180–194. Springer, Heidelberg (2007). doi:10.1007/978-3-540-72734-7_13
6. Demri, S., D'Souza, D., Gascon, R.: Temporal logics of repeating values. J. Log. Comput. **22**(5), 1059–1096 (2012)
7. Demri, S., Figueira, D., Praveen, M.: Reasoning about data repetitions with counter systems. In: LICS, pp. 33–42. IEEE Press (2013)
8. Demri, S., Jurdziński, M., Lachish, O., Lazić, R.: The covering and boundedness problems for branching vector addition systems. J. Comput. Syst. Sci. **79**(1), 23–38 (2013)
9. Demri, S., Lazić, R.: LTL with the freeze quantifier and register automata. ACM Trans. Comput. Log. 10(3) (2009)
10. Emerson, E.A., Halpern, J.Y.: "Sometimes" and "not never" revisited: on branching versus linear time temporal logic. JACM **33**(1), 151–178 (1986)
11. Figueira, D.: Forward-XPath and extended register automata on data-trees. In: ICDT. ACM (2010)
12. Figueira, D.: Alternating register automata on finite data words and trees. Log. Methods Comput. Sci. 8(1) (2012)
13. Figueira, D.: Decidability of downward XPath. ACM Trans. Comput. Log. 13(4) (2012)
14. Figueira, D.: On XPath with transitive axes and data tests. In: PODS, pp. 249–260. ACM (2013)
15. Figueira, D., Figueira, S., Areces, C.: Basic model theory of XPath on data trees. In: ICDT, pp. 50–60. ACM (2014)

16. Figueira, D., Libkin, L.: Pattern logics and auxiliary relations. In: CSL-LICS, pp. 40:1–40:10 (2014)
17. Figueira, D., Segoufin, L.: Future-looking logics on data words and trees. In: Královič, R., Niwiński, D. (eds.) MFCS 2009. LNCS, vol. 5734, pp. 331–343. Springer, Heidelberg (2009). doi:10.1007/978-3-642-03816-7_29
18. Figueira, D., Segoufin, L.: Bottom-up automata on data trees and vertical XPath. In: STACS, vol. 9 of LIPIcs, pp. 93–104. LZI (2011)
19. Jacquemard, F., Segoufin, L., Dimino, J.: FO2($<+1, \sim$) on data trees, data tree automata and branching vector addition systems. Log. Methods Comput. Sci. 12(2) (2016)
20. Jurdziński, M., Lazić, R.: Alternating automata on data trees and XPath satisfiability. ACM Trans. Comput. Log. **12**(3), 19 (2011)
21. Kara, A., Schwentick, T., Zeume, T.: Temporal logics on words with multiple data values. In: FST & TCS (2010)
22. Kupferman, O., Vardi, M.: Memoryful branching-time logic. In: LICS, pp. 265–274. IEEE Press (2006)
23. Lazić, R., Sylvain, S.: Nonelementary complexities for branching VASS, MELL, and extensions. ACM Trans. Comput. Log. **16**(3), 20 (2015)
24. Lisitsa, A., Potapov, I.: Temporal logic with predicate λ-abstraction. In: TIME, pp. 147–155. IEEE Press (2005)
25. Neven, F., Schwentick, T., Vianu, V.: Finite state machines for strings over infinite alphabets. ACM Trans. Comput. Log. **5**(3), 403–435 (2004)
26. Pnueli, A.: The temporal logic of programs. In: FOCS, pp. 46–57. IEEE Press (1977)
27. Rackoff, C.: The covering and boundedness problems for vector addition systems. Theoret. Comput. Sci. **6**(2), 223–231 (1978)
28. 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). doi:10.1007/11874683_3
29. Verma, K.N., Goubault-Larrecq, J.: Karp-Miller trees for a branching extension of VASS. Discrete Math. Theor. Comput. Sci. **7**(1), 217–230 (2005)

Automata, Logic and Formal Languages

Degree of Sequentiality of Weighted Automata

Laure Daviaud[1], Ismaël Jecker[2], Pierre-Alain Reynier[3], and Didier Villevalois[3(✉)]

[1] Warsaw University, Warsaw, Poland
ldaviaud@mimuw.edu.pl
[2] Université Libre de Bruxelles, Brussels, Belgium
ijecker@ulb.ac.be
[3] Aix-Marseille Univ, LIF, CNRS, Marseille, France
{pierre-alain.reynier,didier.villevalois}@lif.univ-mrs.fr

Abstract. Weighted automata (WA) are an important formalism to describe quantitative properties. Obtaining equivalent deterministic machines is a longstanding research problem. In this paper we consider WA with a set semantics, meaning that the semantics is given by the set of weights of accepting runs. We focus on multi-sequential WA that are defined as finite unions of sequential WA. The problem we address is to minimize the size of this union. We call this minimum the degree of sequentiality of (the relation realized by) the WA.

For a given positive integer k, we provide multiple characterizations of relations realized by a union of k sequential WA over an infinitary finitely generated group: a Lipschitz-like machine independent property, a pattern on the automaton (a new twinning property) and a subclass of cost register automata. When possible, we effectively translate a WA into an equivalent union of k sequential WA. We also provide a decision procedure for our twinning property for commutative computable groups thus allowing to compute the degree of sequentiality. Last, we show that these results also hold for word transducers and that the associated decision problem is PSPACE-complete.

1 Introduction

Weighted automata. Finite state automata can be viewed as functions from words to Booleans and, thus, describe languages. Such automata have been extended to define functions from words to various structures yielding a very rich literature, with recent applications in quantitative verification [6]. Weighted

This work has been funded by FNRS, the DeLTA project (ANR-16-CE40-0007), the ARC project *Transform* (Federation Wallonie Brussels), the FNRS CDR project *Flare*, and the PHC project VAST (35961QJ) funded by Campus France and WBI. L. Daviaud was partially supported by ANR Project ELICA ANR-14-CE25-0005, ANR Project RECRE ANR-11-BS02-0010 and by project LIPA that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement Nb 683080).

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 215–230, 2017.
DOI: 10.1007/978-3-662-54458-7_13

automata [18] (WA) is the oldest of such formalisms. They are defined over semirings $(S, \oplus, \otimes)$ by adding weights from S on transitions; the weight of a run is the product of the weights of the transitions, and the weight of a word w is the sum of the weights of the accepting runs on w.

The decidability status of natural decision problems such as universality and equivalence highly depends on the considered semiring [17]. The first operation of the semiring, used to aggregate the values computed by the different runs, plays an important role in the (un)decidability results. Inspired by the setting of word transducers, recent works have considered a set semantics that consists in keeping all these values as a set, instead of aggregating them [11], and proved several decidability results for the resulting class of finite-valued weighted automata [12].

For automata based models, a very important problem is to simplify the models. For instance, deterministic (a.k.a. sequential) machines allow to derive efficient evaluation algorithms. In general, not every WA can be transformed into an equivalent sequential one. The sequentiality problem then asks, given a WA on some semiring $(S, \oplus, \otimes)$, whether there exists an equivalent sequential WA over $(S, \oplus, \otimes)$. This problem ranges from trivial to undecidable, depending on the considered semiring, see [16] for a survey and [14,15] for more recent works.

Sequential transducers. Transducers define rational relations over words. They can be viewed as weighted automata over the semiring of finite sets of words (thus, built over the free monoid); sum is the set union and product is the concatenation extended to sets. When the underlying automaton is deterministic, then the transducer is said to be *sequential*. The class of sequential functions, *i.e.* those realized by sequential transducers, has been characterized among the class of rational functions by Choffrut, see for instance [4] for a presentation:

Theorem 1 ([7]). *Let T be a functional finite state transducer and $[\![T]\!]$ be the function realized by T. The following assertions are equivalent:*

(i) $[\![T]\!]$ satisfies the bounded variation property
(ii) T satisfies the twinning property
(iii) $[\![T]\!]$ is computed by a sequential transducer

In this result, two key tools are introduced: a property of the function, known as the *bounded variation property*, and a pattern property of the transducer, known as the *twinning property*.

Multi-sequential weighted automata. Multi-sequential functions of finite words have been introduced in [8] as those functions that can be realized by a finite union of sequential transducers. A characterization of these functions among the class of rational functions is given in [8]. Recently, this definition has been lifted to relations in [13] where it is proved that the class of so-called multi-sequential relations can be decided in PTIME among the class of rational relations.

We consider in this paper *multi-sequential weighted automata*, defined as finite unions of sequential WA. As described above, and following [11], we consider weighted automata with a set semantics. We argue that multi-sequential

WA are an interesting compromise between sequential and non-deterministic ones. Indeed, sequential WA have a very low expressiveness, while it is in general difficult to have efficient evaluation procedures for non-deterministic WA. Multi-sequential WA allow to encode standard examples requiring non-determinism, yet provide a natural evaluation procedure. Multi-sequential WA can indeed be efficiently evaluated in parallel by using a thread per member of the union, thus avoiding inter-thread communication.

A natural problem consists in minimizing the size of the union of multi-sequential WA that is, given a WA and a natural number k, decide whether it can be realized as a union of k sequential WA. We are also interested in identifying the minimal such k, that we call *degree of sequentiality* of the WA.

Contributions. In this paper, we propose a solution to the problem of the computation of the degree of sequentiality of WA. Following previous works [10,11], we consider WA over infinitary finitely generated groups. We introduce new generalizations of the tools of Choffrut that allow us to characterize the relations that can be defined as unions of k sequential WA: first, a property of relations that extends a Lipschitz property for transducers, and is called *Lipschitz property of order* k (Lip_k for short); second, a pattern property of transducers, called *branching twinning property of order* k (BTP_k for short). We prove:

Theorem 2. *Let W be a weighted automaton with set semantics over an infinitary finitely generated group and k be a positive integer. The following assertions are equivalent:*

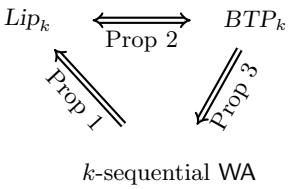
(i) $[\![W]\!]$ *satisfies the Lipschitz property of order* k*,*
(ii) W *satisfies the branching twinning property of order* k*,*
(iii) $[\![W]\!]$ *is computed by a* k*-sequential weighted automaton.*

In addition, the equivalent model of property (iii) can be effectively computed.

As demonstrated by this result, the first important contribution of our work is thus to identify the correct adaptation of the properties of Choffrut suitable to characterize k-sequential relations. Sequential functions are characterized by both a bounded variation and a Lipschitz property [5]. In [10], we introduced a generalization of the bounded variation property to characterize relations that can be expressed using a particular class of cost register automata with exactly k registers, that encompasses the class of k-sequential relations. Though, to characterize k-sequential relations, we here introduce a generalization of the Lipschitz property. We actually believe that this class cannot be characterized by means of a generalization of the bounded variation property. Similarly, the difference between the twinning property of order k introduced in [10] and the branching twinning property of order k introduced in this paper is subtle: we allow here to consider runs on different input words, and the property requires the existence of two runs whose outputs are close on their common input words.

We now discuss the proof of Theorem 2 whose structure is depicted in the picture on the right. In [7], as well as in [10], the difficult part is the construction,

given a machine satisfying the pattern property, of an equivalent deterministic machine. Here again, the most intricate proof of our work is that of Proposition 3: the construction, given a WA satisfying the BTP_k, of an equivalent k-sequential weighted automaton. It is worth noting that it is not a simple extension of [7,10]. Our proof proceeds by induction on k, and the result of [7] constitutes the base case while the tricky part resides in the induction step. Compared with [10], the construction of [10] stores pairwise delays between runs, and picks a minimal subset of "witness" runs that allows to express every other run. In [10], the choice of these witnesses may evolve along an execution while in order to define a k-sequential WA, the way we choose the representative runs should be consistent during the execution. The technical part of our construction is thus the identification of a partition of size at most k of the different runs of the non-deterministic WA such that each element of this partition defines a sequential function. This relies on the branching structure of the twinning property we introduce in this paper.



Our result can also be rephrased in terms of cost register automata [2]. These are deterministic automata equipped with registers that aim to store along the run values from a given semiring S. The restriction of this model to updates of the form $X := X\alpha$ (we say that registers are *independent*) exactly coincides (if we allow k registers) with the class of k-sequential relations. Hence, our result also allows to solve the register minimization problem for this class of CRA.

Beyond weighted automata over infinitary groups, we also prove that our results apply to transducers from A^* to B^*.

Regarding decidability, we show that if the group $\mathbb{G}$ is commutative and has a computable internal operation, then checking whether the BTP_k is satisfied is decidable. As a particular instance of our decision procedure, we obtain that this can be decided in PSPACE for $\mathbb{G} = (\mathbb{Z}, +, 0)$, and show that the problem is PSPACE-hard. Last, we prove that checking the BTP_k for finite-state transducers is also PSPACE-complete.

Organization of the paper. We start with definitions in Sect. 2. In Sect. 3, we introduce our original Lipschitz and branching twinning properties. We present our main construction in Sect. 4. Section 5 is devoted to the presentation of our results about cost register automata, while transducers are dealt with in Sect. 6. Last we present our decidability results and their application to the computation of the degree of sequentiality in Sect. 7. Omitted proofs can be found in [9].

2 Definitions and Examples

Prerequisites and notation. We denote by A a finite alphabet, by A^* the set of finite words on A, by ε the empty word and by $|w|$ the length of a word w. For a set S, we denote by $|S|$ the cardinality of S.

A *monoid* $\mathbb{M} = (M, \otimes, \mathbf{1})$ is a set M equipped with an associative binary operation $\otimes$ with $\mathbf{1}$ as neutral element; the product $\alpha \otimes \beta$ in M may be simply denoted by $\alpha\beta$. If every element of a monoid possesses an inverse - for all $\alpha \in M$, there exists β such that $\alpha\beta = \beta\alpha = \mathbf{1}$ (such a β is unique and is denoted by α^{-1}) - then M is called a group. The monoid (*resp.* group) is said to be *commutative* when $\otimes$ is commutative. Given a finite alphabet B, we denote by $\mathcal{F}(B)$ the free group generated by B.

A *semiring* $\mathbb{S}$ is a set S equipped with two binary operations $\oplus$ (sum) and $\otimes$ (product) such that $(S, \oplus, \mathbf{0})$ is a commutative monoid with neutral element $\mathbf{0}$, $(S, \otimes, \mathbf{1})$ is a monoid with neutral element $\mathbf{1}$, $\mathbf{0}$ is absorbing for $\otimes$ (*i.e.* $\alpha \otimes \mathbf{0} = \mathbf{0} \otimes \alpha = \mathbf{0}$) and $\otimes$ distributes over $\oplus$ (*i.e.* $\alpha \otimes (\beta \oplus \gamma) = (\alpha \otimes \beta) \oplus (\alpha \otimes \gamma)$ and $(\alpha \oplus \beta) \otimes \gamma = (\alpha \otimes \gamma) \oplus (\beta \otimes \gamma)$).

Given a set S, the set of the finite subsets of S is denoted by $\mathcal{P}_{\mathsf{fin}}(S)$. For a monoid $\mathbb{M}$, the set $\mathcal{P}_{\mathsf{fin}}(M)$ equipped with the two operations $\cup$ (union of two sets) and the set extension of $\otimes$ is a semiring denoted $\mathbb{P}_{\mathsf{fin}}(\mathbb{M})$.

From now on, we may identify algebraic structures (monoid, group, semiring) with the set they are defined on when the operations are clear from the context.

Delay and infinitary group. There exists a classical notion of *distance* on words (*i.e.* on the free monoid) measuring their difference: *dist* is defined for any two words u, v as $dist(u, v) = |u| + |v| - 2 * |lcp(u, v)|$ where $lcp(u, v)$ is the longest common prefix of u and v.

When considering a group $\mathbb{G}$ and $\alpha,\ \beta \in \mathbb{G}$, we define the *delay* between α and β as $\alpha^{-1}\beta$, denoted by $delay(\alpha, \beta)$.

Lemma 1. *Given a group $\mathbb{G}$, for all $\alpha, \alpha', \beta, \beta', \gamma, \gamma' \in \mathbb{G}$,*

1. *$delay(\alpha, \beta) = \mathbf{1}$ if and only if $\alpha = \beta$,*
2. *if $delay(\alpha, \alpha') = delay(\beta, \beta')$ then $delay(\alpha\gamma, \alpha'\gamma') = delay(\beta\gamma, \beta'\gamma')$.*

For a finitely generated group $\mathbb{G}$, with a fixed finite set of generators Γ, one can define a distance between two elements derived from the Cayley graph of $(\mathbb{G}, \Gamma)$. We consider here an undirected right Cayley graph: given $\alpha \in \mathbb{G}$, $\beta \in \Gamma$, there is a (non-oriented) edge between α and $\alpha\beta$. Given $\alpha,\ \beta \in \mathbb{G}$, the *Cayley distance* between α and β is the length of the shortest path linking α and β in the undirected right Cayley graph of $(\mathbb{G}, \Gamma)$. It is denoted by $d(\alpha, \beta)$.

For any $\alpha \in \mathbb{G}$, we define the *size* of α (with respect to the set of generators Γ) as the natural number $d(\mathbf{1}, \alpha)$. It is denoted by $|\alpha|$. Note that for a word u, considered as an element of $\mathcal{F}(A)$, the size of u is exactly the length of u (that is why we use the same notation).

Lemma 2. *Given a finitely generated group $\mathbb{G}$ and a finite set of generators Γ, for all $\alpha, \beta \in \mathbb{G}$, $d(\alpha, \beta) = |delay(\alpha, \beta)|$.*

A group $\mathbb{G}$ is said to be *infinitary* if for all $\alpha, \beta, \gamma \in \mathbb{G}$ such that $\alpha\beta\gamma \neq \beta$, the set $\{\alpha^n\beta\gamma^n \mid n \in \mathbb{N}\}$ is infinite. Classical examples of infinite groups such as $(\mathbb{Z}, +, 0)$, $(\mathbb{Q}, \times, 1)$ and the free group generated by a finite alphabet are all infinitary. See [11] for other examples.

Weighted automata. Given a semiring $\mathbb{S}$, weighted automata (WA) are nondeterministic finite automata in which transitions have for weights elements of $\mathbb{S}$. Weighted automata compute functions from the set of words to $\mathbb{S}$: the weight of a run is the product of the weights of the transitions along the run and the weight of a word w is the sum of the weights of the accepting runs labeled by w.

We will consider, for some monoid $\mathbb{M}$, weighted automata over the semiring $\mathbb{P}_{\mathsf{fin}}(\mathbb{M})$. In our settings, instead of considering the semantics of these automata in terms of functions from A^* to $\mathbb{P}_{\mathsf{fin}}(\mathbb{M})$, we will consider it in terms of relations over A^* and $\mathbb{M}$. More precisely, a weighted automaton (with initial and final relations), is formally defined as follows:

Definition 1. *Let A be a finite alphabet, a weighted automaton W over some monoid $\mathbb{M}$ is a tuple $(Q, t_{\mathit{init}}, t_{\mathit{final}}, T)$ where Q is a finite set of states, $t_{\mathit{init}} \subseteq Q \times \mathbb{M}$ (resp. $t_{\mathit{final}} \subseteq Q \times \mathbb{M}$) is the finite initial (resp. final) relation, $T \subseteq Q \times A \times \mathbb{M} \times Q$ is the finite set of transitions.*

A state q is said to be *initial* (*resp. final*) if there is $\alpha \in \mathbb{M}$ such that $(q, \alpha) \in t_{\mathit{init}}$ (*resp.* $(q, \alpha) \in t_{\mathit{final}}$), depicted as $\xrightarrow{\alpha} q$ (resp. $q \xrightarrow{\alpha}$). A run ρ from a state q_1 to a state q_k on a word $w = w_1 \cdots w_k \in A^*$ where for all i, $w_i \in A$, is a sequence of transitions: $(q_1, w_1, \alpha_1, q_2), (q_2, w_2, \alpha_2, q_3), \ldots, (q_k, w_k, \alpha_k, q_{k+1})$. The *output* of such a run is the element of $\mathbb{M}$, $\alpha = \alpha_1\alpha_2 \cdots \alpha_k$. We depict this situation as $q_1 \xrightarrow{w|\alpha} q_{k+1}$. The run ρ is said to be *accepting* if q_1 is initial and q_{k+1} final. This automaton W computes a relation $[\![W]\!] \subseteq A^* \times \mathbb{M}$ defined by the set of pairs $(w, \alpha\beta\gamma)$ such that there are $p, q \in Q$ with $\xrightarrow{\alpha} p \xrightarrow{w|\beta} q \xrightarrow{\gamma}$.

An automaton is *trimmed* if each of its states appears in some accepting run. W.l.o.g., we assume that the automata we consider are trimmed.

Given a weighted automaton $W = (Q, t_{\mathit{init}}, t_{\mathit{final}}, T)$ over some finitely generated group $\mathbb{G}$ with finite set of generators Γ, we define the constant M_W with respect to Γ as $M_W = \max\{|\alpha| \mid (p, a, \alpha, q) \in T \text{ or } (q, \alpha) \in t_{\mathit{init}} \cup t_{\mathit{final}}\}$.

For any positive integer ℓ, a relation $R \subseteq X \times Y$ is said to be *ℓ-valued* if, for all $x \in X$, the set $\{y \mid (x, y) \in R\}$ contains at most ℓ elements. It is said to be *finitely valued* if it is ℓ-valued for some ℓ. A weighted automaton W is said to be *ℓ-valued* (resp. *finite-valued*) if it computes a ℓ-valued (resp. finite-valued) relation.

The union of two weighted automata $W_i = (Q_i, t^i_{\mathit{init}}, t^i_{\mathit{final}}, T_i)$, for $i \in \{1, 2\}$, with disjoint states $Q_1 \cap Q_2 = \varnothing$ is the automaton $W_1 \cup W_2 = (Q_1 \cup Q_2, t^1_{\mathit{init}} \cup t^2_{\mathit{init}}, t^1_{\mathit{final}} \cup t^2_{\mathit{final}}, T_1 \cup T_2)$. States can always be renamed to ensure disjointness. It is trivial to verify that $[\![W_1 \cup W_2]\!] = [\![W_1]\!] \cup [\![W_2]\!]$. This operation can be generalized to the union of k weighted automata.

Definition 2. *A weighted automaton $(Q, t_{\mathit{init}}, t_{\mathit{final}}, T)$ over $\mathbb{M}$ is said to be sequential if $|t_{\mathit{init}}| = 1$ and if for all $p \in Q$, $a \in A$ there is at most one transition in T of the form (p, a, α, q). It is said to be k-sequential if it is a union of k sequential automata. It is said to be multi-sequential if it is k-sequential for some k. A relation is said to be k-sequential (resp. multi-sequential) if it can*

be computed by a k-sequential (resp. multi-sequential) automaton. The degree of sequentiality *of the relation is the minimal k such that it is k-sequential.*

Observe that, unlike the standard definition of sequential weighted automata over $\mathbb{M}$ (see for instance [11]), we allow finite sets of weights to be associated with final states, and not only singletons. This seems more appropriate to us regarding the parallel evaluation model for multi-sequential weighted automata: we prefer to merge threads that only differ by their final outputs. If we define $OutMax = \max_{q \in Q} |\{(q, \alpha) \in t_{final}\}|$, then the standard definition of sequential machines requires $OutMax = 1$. Being k-sequential implies being $(k \cdot OutMax)$-valued. Hence, multi-sequential weighted automata are included in finite-valued ones. However, multi-sequential weighted automata are strictly less expressive than finite-valued ones.

Allowing a final output relation obviously has an impact on the sequentiality degree. We believe that it is possible to fit the usual setting by appropriately reformulating our characterizations. However, this cannot be directly deduced from our current results.

Example 1. Let us consider $A = \{a, b\}$ and $(\mathbb{M}, \otimes, \mathbf{1}) = (\mathbb{Z}, +, 0)$. The weighted automaton W_0 given in Fig. 1(a) computes the function f_{last} that associates with a word wa (resp. wb) its number of occurrences of the letter a (resp. b), and associates 0 with the empty word. It is easy to verify that the degree of sequentiality of f_{last} is 2. It is also standard that the function f^*_{last} mapping the word $u_1\# \dots \#u_n$ (for any n) to $f_{last}(u_1) + \dots + f_{last}(u_n)$ is not multi-sequential (see for instance [13]) whereas it is single-valued.

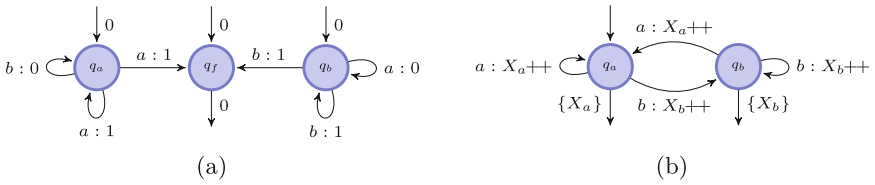


Fig. 1. (a) Example of a weighted automaton W_0 computing the function f_{last}. (b) Example of a cost register automaton C_0 computing the function f_{last}. The updates are abbreviated: $X_a{+}{+}$ means both $X_a := X_a + 1$ and $X_b := X_b$ (and conversely).

3 Lipschitz and Branching Twinning Properties

Sequential transducers have been characterized in [7] by Choffrut by means of a so-called bounded-variation property and a twinning property. The bounded-variation property is actually equivalent to a Lipschitz-like property (see for instance [5]). We provide adaptations of the Lipschitz and twinning properties so as to characterize k-sequential WA.

We consider a finitely generated infinitary group $\mathbb{G}$ and we fix a finite set of generators Γ.

3.1 Lipschitz Property of Order k

Given a partial mapping $f : A^* \rightharpoonup B^*$, the Lipschitz property states that there exists $L \in \mathbb{N}$ such that for all $w, w' \in A^*$ such that $f(w), f(w')$ are defined, we have $dist(f(w), f(w')) \leqslant L\, dist(w, w')$ (see [5]). Intuitively, this property states that, for two words, their images by f differ proportionally to those words. This corresponds to the intuition that the function can be expressed by means of a sequential automaton.

When lifting this property to functions that can be expressed using a k-sequential automaton, we consider $k+1$ input words and require that two of those must have proportionally close images by f. The extension to relations $R \subseteq A^* \times B^*$ requires that for all $k+1$ pairs chosen in R, two of those have their range components proportionally close to their domain components. In addition, for relations, an input word may have more than one output word, we thus need to add a constant 1 in the right-hand side. Finally, our framework is that of infinitary finitely generated groups. Instead of $dist(,)$, we use the Cayley distance $d(,)$ to compare elements in the range of the relation.

Definition 3. *A relation $R \subseteq A^* \times \mathbb{G}$ satisfies the Lipschitz property of order k if there is a natural L such that for all pairs $(w_0, \alpha_0), \ldots, (w_k, \alpha_k) \in R$, there are two indices i, j such that $0 \leqslant i < j \leqslant k$ and $d(\alpha_i, \alpha_j) \leqslant L\,(dist(w_i, w_j) + 1)$.*

Example 2. The group $(\mathbb{Z}, +, 0)$ is finitely generated with $\{1\}$ as a set of generators. The function f_{last} does not satisfy the Lipschitz property of order 1 (take $w_1 = a^N a$ and $w_2 = a^N b$), but it satisfies the Lipschitz property of order 2.

Using the pigeon hole principle, it is easy to prove the implication from (*iii*) to (*i*) of Theorem 2:

Proposition 1. *A k-sequential relation satisfies the Lipschitz property of order k.*

3.2 Branching Twinning Property of Order k

The idea behind the branching twinning property of order k is to consider $k+1$ runs labeled by arbitrary words with k cycles. If the branching twinning property is satisfied then there are two runs among these $k+1$ such that the values remain close (i.e. the Cayley distance between these values is bounded) along the prefix part of these two runs that read the same input. This property is named after the intuition that the $k+1$ runs can be organized in a tree structure where the prefixes of any two runs are on the same branch up to the point where those two runs do not read the same input anymore.

Definition 4. *A weighted automaton over $\mathbb{G}$ satisfies the branching twinning property of order k (denoted by BTP_k) if: (see Fig. 2)*

- *for all states $\{q_{i,j} \mid i, j \in \{0, \ldots, k\}\}$ with $q_{0,j}$ initial for all j,*
- *for all γ_j such that $(q_{0,j}, \gamma_j) \in t_{init}$ with $j \in \{0, \ldots, k\}$,*

– *for all words* $u_{i,j}$ *and* $v_{i,j}$ *with* $1 \leqslant i \leqslant k$ *and* $0 \leqslant j \leqslant k$ *such that there are* $k+1$ *runs satisfying for all* $0 \leqslant j \leqslant k$, *for all* $1 \leqslant i \leqslant k$, $q_{i-1,j} \xrightarrow{u_{i,j}|\alpha_{i,j}} q_{i,j}$ *and* $q_{i,j} \xrightarrow{v_{i,j}|\beta_{i,j}} q_{i,j}$,

there are $j \neq j'$ *such that for all* $i \in \{1, \ldots, k\}$, *if for every* $1 \leqslant i' \leqslant i$, *we have* $u_{i',j} = u_{i',j'}$ *and* $v_{i',j} = v_{i',j'}$, *then we have*

$$delay(\gamma_j \alpha_{1,j} \cdots \alpha_{i,j}, \gamma_{j'} \alpha_{1,j'} \cdots \alpha_{i,j'}) = delay(\gamma_j \alpha_{1,j} \cdots \alpha_{i,j} \beta_{i,j}, \gamma_{j'} \alpha_{1,j'} \cdots \alpha_{i,j'} \beta_{i,j'}).$$

Example 3. The weighted automaton W_0, given in Fig. 1(a), does not satisfy the BTP_1 (considering loops around q_a and q_b). One can prove however that it satisfies the BTP_2.

Let us denote by W_1 the weighted automaton obtained by concatenating W_0 with itself, with a fresh # separator letter. W_1 realizes the function f^2_{last} defined as $f^2_{last}(u\#v) = f_{last}(u) + f_{last}(v)$. We can see that the minimal k such that W_1 satisfies the BTP_k is $k = 4$. As we will see, this is the sequentiality degree of f^2_{last}.

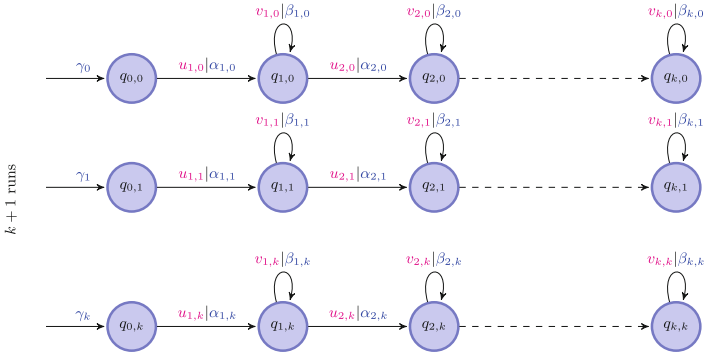


Fig. 2. Branching twinning property of order k

3.3 Equivalence of Lipschitz and Branching Twinning Properties

We can prove that a weighted automaton satisfies the BTP_k if and only if its semantics satisfies the Lipschitz property of order k. This implies that the branching twinning property of order k is a machine independent property, *i.e.* given two WA W_1, W_2 such that $[\![W_1]\!] = [\![W_2]\!]$, W_1 satisfies the BTP_k iff W_2 satisfies the BTP_k.

Proposition 2. *A weighted automaton* W *over an infinitary finitely generated group* $\mathbb{G}$ *satisfies* BTP_k *if and only if* $[\![W]\!]$ *satisfies the Lipschitz property of order* k.

Proof (Sketch). Let us sketch the proof of the Proposition. First, suppose that W does not satisfy the BTP_k. Then consider a witness of this non satisfaction. Fix an integer L. By pumping the loops in this witness (enough time and going backward), one can construct $k+1$ words that remain pairwise sufficiently close while their outputs are pairwise at least at distance L. This leads to prove that $[\![W]\!]$ does not satisfy the Lipschitz property of order k.

Conversely, consider that the BTP_k is satisfied. For all $k+1$ pairs of words and weights in $[\![W]\!]$, we have $k+1$ corresponding runs in W labeled by those words. By exhibiting cycles on these runs, we can get an instance of BTP_k as in Fig. 2 such that the non-cycling part is bounded (in length). By BTP_k, there are two runs that have the same delays before and after the loops appearing in their common prefix. Thus, we can bound the distance between the two weights produced by those runs proportionally to the distance between the two input words, proving that the Lipschitz property is satisfied. □

4 Constructing a *k*-sequential Weighted Automaton

As explained in the introduction, the most intricate part in the proof of Theorem 2 is to prove that (*ii*) implies (*iii*). We give a constructive proof of this fact as stated in the following proposition.

Proposition 3. *Given a weighted automaton W satisfying the BTP_k, one can effectively build k sequential weighted automata whose union is equivalent to W.*

Let $W = (Q, t_{init}, t_{final}, T)$ be a weighted automaton that satisfies the BTP_k. The construction is done in two steps. First, we build an infinite sequential weighted automaton D_W equivalent to W, using the subset construction with delays presented in [4]. Then, by replacing infinite parts of D_W with finite automata, we build k sequential weighted automata whose union is equivalent to W.

Let us sketch the main ideas behind the construction of D_W. The states of D_W are the subsets S of $Q \times \mathbb{G}$. On input $u \in A^*$, D_W selects an initial run $\rho : \xrightarrow{\alpha_0} p_0 \xrightarrow{u|\alpha} p$ of W, outputs the corresponding $\alpha \in \mathbb{G}$, and, in order to keep track of all the runs $\rho' : \xrightarrow{\beta_0} q_0 \xrightarrow{u|\beta} q$ of W over the input u, stores in its state the corresponding pairs $(q', delay(\alpha_0\alpha, \beta_0\beta))$. The detailed construction, together with the proofs of its properties, adapted from [4] to fit our settings, can be found in [9].

If W is a transducer, i.e., a weighted automaton with weights in a free monoid, and W satisfies the BTP_1, which is equivalent to the twinning property, Lemma 17 of [4] proves that the trim part of D_W is finite. This lemma can be generalized to any kind of weighted automata, proving our proposition in the particular case $k = 1$. Let us now prove the general result by induction. Suppose that $k > 1$, and that the proposition is true for every integer strictly smaller than k. We begin by exposing two properties satisfied by D_W.

Since W satisfies the BTP_k, it also satisfies the notion of TP_k introduced in [10], and, by Proposition 1 of that paper, it is ℓ-valued for some integer ℓ

effectively computable. Let $N_W = 2M_W|Q|^{\ell|Q|}$, let $S \in Q \times \mathbb{G}$ be a state of the trim part of D_W, and let $W_S = (Q, S, t_{final}, T)$ be the weighted automaton obtained by replacing the initial output relation of W with S. The following properties are satisfied.

$\mathbf{P}_1$: The size of S is bounded by $\ell|Q|$;
$\mathbf{P}_2$: If there exists a pair $(q, \alpha) \in S$ such that $|\alpha| > N_W$, $[\![W_S]\!]$ is k-sequential.

The proof of $\mathbf{P}_1$ follows from the ℓ-valuedness of W. The main difficulty of the demonstration of Proposition 3 lies in the proof of $\mathbf{P}_2$, which can be sketched as follows. Using the fact that there exists $(q, \alpha) \in S$ such that $|\alpha| > N_W$, we expose a partition of S into two subsets S' and S'' satisfying the $BTP_{k'}$, respectively the $BTP_{k''}$, for some $1 \leqslant k', k'' < k$ such that $k' + k'' \leqslant k$. This is proved by using the fact that W satisfies the BTP_k, and that the branching nature of the BTP allows us to combine unsatisfied instances of the BTP over $W_{S'}$ and $W_{S''}$ to build unsatisfied instances of the BTP over W. Then, since $k' < k$ and $k'' < k$, $[\![S']\!]$ is k'-sequential and $[\![S'']\!]$ is k''-sequential by the induction hypothesis. Finally, as S is the union of S' and S'', W_S is equivalent to the union of $W_{S'}$ and $W_{S''}$, and $\mathbf{P}_2$ follows, since $k' + k'' \leqslant k$.

The properties $\mathbf{P}_1$ and $\mathbf{P}_2$ allow us to expose k sequential weighted automata $\overline{V}_1, \ldots, \overline{V}_k$ whose union is equivalent to W. Let U denote the set containing the accessible states S of D_W that contain only pairs (q, α) satisfying $|\alpha| \leqslant N_W$. As there are only finitely many $\alpha \in \mathbb{G}$ such that $|\alpha| \leqslant N_W$, $\mathbf{P}_1$ implies that U is finite. Moreover, as a consequence of $\mathbf{P}_2$, for every state $S \notin U$ in the trim part of D_W, W_S can be expressed as the union of k sequential weighted automata $V_i(S)$, with $1 \leqslant i \leqslant k$. For every $1 \leqslant i \leqslant k$, let $\overline{V}_i$ be the sequential weighted automaton that copies the behaviour of D_W as long as the latter stays in U, and swaps to $V_i(S)$ as soon as D_W enters a state $S \notin U$. Then D_W is equivalent to the union of the $\overline{V}_i$, $1 \leqslant i \leqslant k$, which proves the desired result, since D_W is equivalent to W. The detailed proofs can be found in [9].

5 Cost Register Automata with Independent Registers

Recently, a new model of machine, named cost register automata (CRA), has been introduced in [2]. We present in this section how the class of k-sequential relations is also characterized by a specific subclass of cost register automata.

A cost register automaton (CRA) [2] is a deterministic automaton with registers containing values from a set S and that are updated through the transitions: for each register, its new value is computed from the old ones and from elements of S combined using some operations over S. The output value is computed from the values taken by the registers at the end of the processing of the input. Hence, a CRA defines a relation in $A^* \times S$.

In this paper, we focus on a particular structure $(\mathbb{M}, \otimes c)$ defined over a monoid $(\mathbb{M}, \otimes, \mathbf{1})$. In such a structure, the only updates are unary and are of the form $X := Y \otimes c$, where $c \in \mathbb{M}$ and X, Y are registers. When $\mathbb{M}$ is $(\mathbb{Z}, +, 0)$, this class of automata is called additive cost register automata [3]. When $\mathbb{M}$ is the

free monoid $(A^*, ., \varepsilon)$, this class is a subclass of streaming string transducers [1] and turns out to be equivalent to the class of rational functions on words, *i.e.* those realized by finite-state transducers.

While cost register automata introduced in [2] define functions from A^* to $\mathbb{M}$, we are interested in defining finite-valued relations. To this aim, we slightly modify the definition of CRA, allowing to produce a set of values computed from register contents.

Definition 5. *A cost register automaton on the alphabet A over the monoid $(\mathbb{M}, \otimes, \mathbf{1})$ is a tuple $(Q, q_{init}, \mathcal{X}, \delta, \mu)$ where Q is a finite set of states, $q_{init} \in Q$ is the initial state and $\mathcal{X}$ is a finite set of registers. The transitions are given by the function $\delta : Q \times A \to (Q \times \mathcal{UP}(\mathcal{X}))$ where $\mathcal{UP}(\mathcal{X})$ is the set of functions $\mathcal{X} \to \mathcal{X} \times \mathbb{M}$ that represents the updates on the registers. Finally, μ is a finite set of $Q \times \mathcal{X} \times \mathbb{M}$ (the output relation).*

The semantics of such an automaton is as follows: if an update function f labels a transition and $f(Y) = (X, \alpha)$, then the register Y after the transition will take the value $\beta\alpha$ where β is the value contained in the register X before the transition. More precisely, a valuation ν is a mapping from $\mathcal{X}$ to $\mathbb{M}$ and let $\mathcal{V}$ be the set of such valuations. The initial valuation ν_{init} is the function associating with each register the value $\mathbf{1}$. A configuration is an element of $Q \times \mathcal{V}$. The initial configuration is (q_{init}, ν_{init}). A run on a word $w = w_1 \cdots w_k \in A^*$ where for all i, $w_i \in A$, is a sequence of configurations $(q_1, \nu_1)(q_2, \nu_2) \dots (q_{k+1}, \nu_{k+1})$ satisfying that for all $1 \leqslant i \leqslant k$, and all registers Y, if $\delta(q_i, w_i) = (q_{i+1}, g_i)$ with $g_i(Y) = (X, \alpha)$, then $\nu_{i+1}(Y) = \nu_i(X)\alpha$. Moreover, the run is said to be accepting if (q_1, ν_1) is the initial configuration and there are X, α such that $(q_{k+1}, X, \alpha) \in \mu$.

A cost register automaton C computes a relation $[\![C]\!] \subseteq A^* \times \mathbb{M}$ defined by the set of pairs $(w, \nu_{k+1}(X)\alpha)$ such that $(q_1, \nu_1)(q_2, \nu_2) \dots (q_{k+1}, \nu_{k+1})$ is an accepting run of C on w and $(q_{k+1}, X, \alpha) \in \mu$.

Definition 6. *A cost register automaton is said to be with independent registers if for any update function f labelling a transition, $f(Y) = (X, \alpha)$ implies $X = Y$.*

Example 4. Consider $A = \{a, b\}$ and $(\mathbb{M}, \otimes, \mathbf{1}) = (\mathbb{Z}, +, 0)$. The cost register automaton C_0 given in Fig. 1(b) computes the function f_{last} introduced in Example 1. The register X_a (*resp.* X_b) stores the number of occurrences of the letter a (*resp.* b). Observe that these two registers are independent.

Independence of registers is tightly related to sequentiality of WA. We prove:

Proposition 4. *For all positive integers k, a relation is k-sequential if and only if it is computed by a cost register automaton with k independent registers.*

CRA are deterministic by definition, and a challenging minimisation problem is captured by the notion of *register complexity*. It is defined for a relation as the minimal integer k such that it can be defined by a CRA with k registers. By Proposition 4, results on the computation of the degree of sequentiality presented

in Sect. 7 thus also allow to compute the register complexity for CRA with independent registers.

One can also show that the class of CRA with k independent registers is equivalent to the class of CRA with k registers, updates of the form $X := Y\alpha$, and that are copyless (every register appears at most once in the right-hand side of an update function).

The class of CRA with k non-independent registers was characterized in [10] using the twinning property of order k. This property is weaker than our branching twinning property of order k as it requires the same conclusion but only for runs labeled by the same input words.

6 The Case of Transducers

A transducer is defined as a weighted automaton with weights in the monoid B^*. It can thus be seen as a weighted automaton with weights in the free group $\mathcal{F}(B)$. We say that a transducer T satisfies the branching twinning property of order k if, viewed as a weighted automaton over $\mathcal{F}(B)$, it satisfies the BTP_k. Similarly, a relation $R \subseteq A^* \times B^*$ is said to satisfy the Lipschitz property of order k iff it is the case when viewing R as a relation in $A^* \times \mathcal{F}(B)$.

A relation R of $A^* \times B^*$ is said to be *positive k-sequential* if it is computed by a k-sequential weighted automaton with weights in B^* (weights on the transitions in B^* and initial and final relations in $Q \times B^*$ where Q is its set of states). As for the general case, it is easy to see that a relation is positive k-sequential if and only if it is computed by a cost register automaton with k independent registers, with updates of the form $X := Xc$ where $c \in B^*$ and with an output relation $\mu \subseteq Q \times \mathcal{X} \times B^*$.

Theorem 3. *Let T be a transducer from A^* to B^*, and k be a positive integer. The following assertions are equivalent:*

(i) $[\![T]\!]$ satisfies the Lipschitz property of order k,
(ii) T satisfies the branching twinning property of order k,
(iii) $[\![T]\!]$ is positive k-sequential.

The assertions (*i*) and (*ii*) are equivalent by Theorem 2. The fact that the assertion (*iii*) implies the assertion (*ii*) is also a consequence of Theorem 2 and of the fact that the branching twinning property of order k is a machine-independent characterization. Finally, it remains to prove that the assertion (*ii*) implies the assertion (*iii*).

By hypothesis, $[\![T]\!] \subseteq A^* \times B^*$ is computed by a transducer that satisfies the branching twinning property of order k. Thus, by Theorem 2, it is computed by a CRA over $\mathcal{F}(B)$ with k independent registers. We conclude using the:

Proposition 5. *A relation in $A^* \times B^*$ is computed by a cost register automaton over $\mathcal{F}(B)$ with k independent registers if and only if it is computed by a cost register automaton over B^* with k independent registers.*

7 Decidability of BTP_k and Computation of the Sequentiality Degree

In this section, we prove the decidability of the following problem under some hypotheses on the group $\mathbb{G}$:

The BTP_k Problem: given a weighted automaton W over some group $\mathbb{G}$ and a number k, does W satisfy the BTP_k?

As a corollary of Theorem 2, this allows to compute the degree of sequentiality for weighted automata. We will consider two settings: first weighted automata over some computable commutative group and second, word transducers.

Our decision procedures non-deterministically guess a counter-example to the BTP_k. First, we show that if there exists such a counter-example with more than k loops, then there exists one with k loops. For simplicity, we can assume that the counter-example contains $k(k+1)/2$ loops *i.e.* exactly one loop per pair (j, j'), with $0 \leqslant j < j' \leqslant k$. This allows the procedure to first guess the "skeleton" of the counter example, and then check that this skeleton can be turned into a real counter-example. The skeleton consists of the vectors of states, and, for each pair (j, j') of run indices, indicates the index $\chi(j, j')$ of the last loop such that input words of runs j and j' are equal up to this loop, and the index $\eta(j, j')$ of the loop that induces a different delay (with $\eta(j, j') \leqslant \chi(j, j')$).

Case of computable commutative groups. We write $W = (Q, t_{init}, t_{final}, T)$ and let $n = |Q|$. In order to decide the branching twinning property, we will consider the $k+1$-th power of W, denoted W^{k+1}, which accepts the set of $k+1$ synchronized runs in W. We write its runs as $\boldsymbol{\rho} = (\rho_i)_{0 \leqslant i \leqslant k}$ and denote by α_i the weight of run ρ_i.

Theorem 4. *Let $\mathbb{G} = (G, \otimes)$ be a commutative group such that the operation $\otimes$ and the equality check are computable. Then the BTP_k problem is decidable.*

Proof (Sketch). It is easy to observe that for commutative groups, the constraint expressed on the delay in the BTP_k boils down to checking that loops have different weights.

The procedure first guesses the skeleton of a counter-example as explained above. The procedure then non-deterministically verifies that the skeleton can be completed into a concrete counter-example. To this end, it uses the information stored in this skeleton about how input words are shared between runs (indices $\chi(j, j')$) to identify the power $p \leqslant k+1$ of W in which the run should be identified. The procedure is based on the two following subroutines:

- first, given two vectors of states $v, v' \in Q^p$, checking that there exists a path from v to v' in W^p is decidable,
- second, the following problem is decidable: given a vector of states $v \in Q^p$ and a pair $1 \leqslant j \neq j' \leqslant p$, check that there exists a cycle $\boldsymbol{\rho}$ around v in W^p such that $delay(\alpha_j, \alpha_{j'}) \neq \mathbf{1}$. The procedure non-deterministically guesses the cycle in W^p (its length can be bounded by $2n^p$) and computes incrementally the value of $delay(\alpha_j, \alpha_{j'})$. □

If we consider the group $(\mathbb{Z}, +)$, we can verify that the above procedure runs in PSPACE if k is given in unary. In addition, using ideas similar to a lower bound proved in [3], we can reduce the emptiness of k deterministic finite state automata to the BTP_k problem, yielding:

Theorem 5. *Over $(\mathbb{Z}, +)$, the BTP_k problem is* PSPACE*-complete (k in unary).*

Case of transducers. For word transducers, the authors of [19] prove that a counter-example to the (classical) twinning property is either such that loops have output words of different length, or such that output words produced on the runs leading to the loops have a mismatch.

Inspired by this result, we show that the skeleton described above can be enriched with the information, for each pair of run indices (j, j'), whether one should look for a loop whose output words have distinct lengths, or for a mismatch on the paths leading to the loop. These different properties can all be checked in PSPACE, yielding:

Theorem 6. *Over $(B^*, \cdot)$, the BTP_k problem is* PSPACE*-complete (k in unary)*[1].

8 Conclusion

Multi-sequential machines are an interesting compromise between sequential and finite-valued ones. This yields the natural problem of the minimization of the size of the union. In this paper, we have solved this problem for weighted automata over an infinitary finitely generated group, a setting that encompasses standard groups. To this end, we have introduced a new twinning property, as well as a new Lipschitz property, and have provided an original construction from weighted automata to k-sequential weighted automata, extending the standard determinization of transducers in an intricate way. In addition, the characterization by means of a twinning property allows to derive efficient decision procedures, and all our results are also valid for word transducers.

As a complement, these results can be generalized to non finitely generated groups, using ideas similar to those developed in [10]. As future work, we plan to lift these results to other settings, like infinite or nested words. Another challenging research direction consists in considering other operations to aggregate weights of runs.

References

1. Alur, R., Cerný, P.: Expressiveness of streaming string transducers. In: IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, 15–18 December 2010, Chennai, India. LIPIcs, vol. 8, pp. 1–12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2010)

[1] The transducer is viewed as a weighted automaton over $\mathcal{F}(B)$.

2. Alur, R., D'Antoni, L., Deshmukh, J.V., Raghothaman, M., Yuan, Y.: Regular functions and cost register automata. In: 28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, 25–28 June 2013, pp. 13–22. IEEE Computer Society (2013)
3. Alur, R., Raghothaman, M.: Decision problems for additive regular functions. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013. LNCS, vol. 7966, pp. 37–48. Springer, Heidelberg (2013). doi:10.1007/978-3-642-39212-2_7
4. Béal, M., Carton, O.: Determinization of transducers over finite and infinite words. Theor. Comput. Sci. **289**(1), 225–251 (2002). http://dx.doi.org/10.1016/S0304-3975(01)00271-7
5. Berstel, J.: Transductions and Context-free Languages. Springer, Heidelberg (2013)
6. Chatterjee, K., Doyen, L., Henzinger, T.A.: Quantitative languages. ACM Trans. Comput. Log. **11**(4) (2010). http://doi.acm.org/10.1145/1805950.1805953
7. Choffrut, C.: Une caracterisation des fonctions sequentielles et des fonctions sous-sequentielles en tant que relations rationnelles. Theor. Comput. Sci. **5**(3), 325–337 (1977). http://dx.doi.org/10.1016/0304-3975(77)90049-4
8. Choffrut, C., Schutzenberger, M.P.: Decomposition de fonctions rationnelles. In: Monien, B., Vidal-Naquet, G. (eds.) STACS 1986. LNCS, vol. 210, pp. 213–226. Springer, Heidelberg (1986). doi:10.1007/3-540-16078-7_78
9. Daviaud, L., Jecker, I., Reynier, P.A., Villevalois, D.: Degree of sequentiality of weighted automata. Research Report 1701.04632, arXiv, Jan 2017. http://arxiv.org/abs/1701.04632
10. Daviaud, L., Reynier, P.A., Talbot, J.M.: A generalised twinning property for minimisation of cost register automata. In: 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2016. IEEE Computer Society (2016, to appear)
11. Filiot, E., Gentilini, R., Raskin, J.F.: Quantitative languages defined by functional automata. Logical Methods Comput. Sci. **11**(3:14), 1–32 (2015). http://arxiv.org/abs/0902.3958
12. Filiot, E., Gentilini, R., Raskin, J.: Finite-valued weighted automata. In: 34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, 15–17 December 2014, New Delhi, India. LIPIcs, vol. 29, pp. 133–145. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2014)
13. Jecker, I., Filiot, E.: Multi-sequential word relations. In: Potapov, I. (ed.) DLT 2015. LNCS, vol. 9168, pp. 288–299. Springer, Cham (2015). doi:10.1007/978-3-319-21500-6_23
14. Kirsten, D.: Decidability, undecidability, and pspace-completeness of the twins property in the tropical semiring. Theor. Comput. Sci. **420**, 56–63 (2012). http://dx.doi.org/10.1016/j.tcs.2011.11.006
15. Kirsten, D., Lombardy, S.: Deciding unambiguity and sequentiality of polynomially ambiguous min-plus automata. In: 26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, 26–28 February 2009, Freiburg, Germany, Proceedings. LIPIcs, vol. 3, pp. 589–600. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany (2009)
16. Lombardy, S., Sakarovitch, J.: Sequential? Theor. Comput. Sci. **356**(1–2), 224–244 (2006). http://dx.doi.org/10.1016/j.tcs.2006.01.028
17. Sakarovitch, J.: Elements of Automata Theory. Cambridge University Press (2009). http://www.cambridge.org/uk/catalogue/catalogue.asp?isbn=9780521844253
18. Schützenberger, M.P.: On the definition of a family of automata. Inf. Control **4**, 245–270 (1961)
19. Weber, A., Klemm, R.: Economy of description for single-valued transducers. Inf. Comput. **118**(2), 327–340 (1995). http://dx.doi.org/10.1006/inco.1995.1071

Emptiness Under Isolation and the Value Problem for Hierarchical Probabilistic Automata

Rohit Chadha[1(✉)], A. Prasad Sistla[2], and Mahesh Viswanathan[3]

[1] University of Missouri, Columbia, USA
chadhar@missouri.edu
[2] University of Illinois, Chicago, USA
sistla@uic.edu
[3] University of Illinois at Urbana-Champaign, Urbana, USA
vmahesh@illinois.edu

Abstract. k-Hierarchical probabilistic automata (k-HPA) are probabilistic automata whose states are stratified into $k+1$ levels such that from any state, on any input symbol, at most one successor belongs to the same level, while the remaining belong to higher levels. Our main result shows that the emptiness and universality problems are decidable for k-HPAs with isolated cut-points; recall that a cut-point x is isolated if the acceptance probability of every word is bounded away from x. Our algorithm for establishing this result relies on computing an approximation of the value of an HPA; the value of a probabilistic automaton is the supremum of the acceptance probabilities of all words. Computing the exact value of a probabilistic automaton is an equally important problem and we show that the problem is **co-R.E.**-complete for k-HPAs, for $k \geq 2$ (as opposed to $\mathbf{\Pi_2^0}$-complete for general probabilistic automata). On the other hand, we also show that for 1-HPAs the value can be computed in exponential time.

1 Introduction

k-Hierarchical probabilistic automata (HPAs) [12] are a syntactic sub-class of probabilistic automata, whose states are stratified into $k+1$ levels. Like probabilistic automata, the next state on an input symbol is determined stochastically. However, transitions are required to "respect levels" — from any state q, on any input symbol a, at most one possible next state belongs to the same level as q, with the others being constrained to belong to levels higher than q's. Such automata can recognize languages over finite (hierarchical probabilistic finite automata) or infinite words (hierarchical probabilistic Muller automata) depending on the notion of accepting runs. Given a threshold x, the language recognized by an HPA $\mathcal{A}$ is the collection of all input strings such that the measure of all accepting runs on the input is $> x$.

HPAs arise naturally as models of client-server systems with stochastic server failures, concurrent systems under probabilistic context-bounded schedulers, and business enterprise systems [2] and these have been analyzed using automated

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 231–247, 2017.
DOI: 10.1007/978-3-662-54458-7_14

tools for HPAs [4]. HPAs were introduced in [12] as a computationally tractable subclass of probabilistic automata. When the acceptance threshold is extremal, i.e., 0 or 1, many verification problems for HPAs become decidable. While (general) probabilistic Büchi automata can recognize non-regular languages with acceptance threshold 0 and 1, HPAs were shown to recognize only regular languages [12]. Classical (qualitative) verification questions like emptiness and universality are decidable in low complexity classes (**NL** and **PSPACE**). In contrast, the emptiness problem for probabilistic Büchi automata with threshold 0 is undecidable [1] and $\mathbf{\Pi_2^0}$-complete [9,12].

Surprisingly, however, the landscape changes completely when the threshold is taken to be $x \in (0,1)$. Even 1-HPAs[1] can recognize non-regular languages when the acceptance threshold is $\frac{1}{2}$ [11]. Though emptiness and universality problems are decidable for 1-HPAs [11], these problems are undecidable for 2-HPAs (and higher) [4]. In this paper, we present results that support the thesis that, despite the many negative results about HPAs in [4,8], HPAs are indeed a computationally tractable model of open probabilistic systems. Specifically, we present results that show that the value of HPAs can be approximated to a given degree of precision ϵ unlike general probabilistic automata. Hence HPAs can be "approximately verified"; we can declare the language of a PFA to be non-empty/empty if the value of the HPA is at least ϵ more/less than the threshold.

The main results in this paper pertain to HPAs with isolated cut-points and the value problem for HPAs. A threshold x is said to be isolated for a probabilistic automaton (not necessarily hierarchical) $\mathcal{A}$, if there is an $\epsilon > 0$ such that the acceptance probability of any word is either at most $x - \epsilon$ or at least $x + \epsilon$, i.e., the probability of acceptance of any word is bounded away from x. Automata with isolated cut-points describe algorithms to which algorithmic techniques like amplification can be applied, and are constant space analogs of complexity classes **BPP** and **RP**. An important classical result due to Rabin [19] is that though probabilistic automata over finite words (PFA) can recognize non-regular languages when the threshold $x \in (0,1)$, they only recognize regular languages when x is isolated. The extension of this result to automata on infinite words is not known. In this paper, we show that HPAs on infinite words with isolated cut-points recognize ω-regular languages.

Even though probabilistic finite automata (PFAs) with isolated cut-points recognize regular languages, it is not known if the following problem is decidable: Given a PFA with an isolated cut-point x, determine if some input string is accepted with probability $> x$. Our main result is that for HPAs with isolated cut-points, this emptiness problem is decidable. Our result applies to both HPAs over finite words and HPAs over infinite words. In fact, we show that checking if an HPA's (with isolated cut-point) language is equal to any given regular language is decidable; thus, even checking universality is decidable.

Our proof for the decidability of emptiness under isolation is based on solving another classical problem for probabilistic automata, namely, computing the

[1] 0-HPAs are just deterministic machines. Thus, 1-HPAs are automata with fewest number of levels that have some stochastic behavior.

value of an automaton. The value of an automaton is the *least upper bound* of the acceptance probabilities of all words. The decision version of the value problem is known to undecidable [5,13,16]. We show that for HPAs (over finite or infinite words) the value can be *approximated* to precision γ ($\gamma < 1$) in time that is doubly exponential in the size of the automaton and exponential in $\mathsf{poly}(\log(\frac{1}{\gamma}))$. The approximation algorithm is obtained by observing that in an HPA, for any finite word v, there is a "short" word u such that the distribution on states after u is very "close" to the distribution after input v; the length of u only depends on the size of the automaton and the approximation factor. Thus to approximate the value of an HPA up-to γ, we compute the maximum of the acceptance probabilities of all "short" words. Having an algorithm to approximate the value of an automaton immediately gives us an algorithm for checking language emptiness of an HPA $\mathcal{A}$ with isolated cut-point x as follows. We progressively compute the value of $\mathcal{A}$ with increasing precision. Suppose at some point the value is approximated by v with precision γ. If $v - \gamma > x$ then we know that $\mathcal{A}$ has a non-empty language. On the other hand, if $v + \gamma < x$ then $\mathcal{A}$'s language is empty. Since x is isolated, we are guaranteed that eventually the precision γ is low enough to ensure that one of these two conditions hold.

In addition to the algorithm to approximate the value of an HPA, we characterize the precise complexity of the value problem for HPAs (on both finite and infinite words). We show that the value problem is in **EXPTIME** for 1-HPAs, as follows. First, we prove that the value of a 1-HPA is a fraction whose size (i.e., the length of the binary representation of its denominator) is at most exponential in the number of states of the automaton. Then, we present an algorithm that computes the value exactly, by employing binary search on rationals [20], together with an algorithm for emptiness checking for 1-HPA [11]. We show that the value problem is **co-R.E.**-complete for 2-HPAs (and higher). In contrast, for general PFAs, the value problem is known to be $\mathbf{\Pi_2^0}$-complete [13]. Finally, we also show that the problem of checking if a cut-point x is not isolated for a probabilistic automaton $\mathcal{A}$ can be reduced in polynomial time to the value problem. This, along with the results for the value problem for HPAs, shows that the problem of checking isolation is **R.E.**-complete for 2-HPAs (and higher), and is in **co-R.E.** for 1-level HPA.

The paper is organized as follows. We describe closely related work next. Section 2 contains notations and definitions. Section 3 has definitions of HPAs and results on the regularity of the language under isolated cut points. Section 4 has results on the approximation of HPAs and decidability of emptiness under isolation. Section 5 has the results for the value problem and isolated cut point problem. Section 6 has conclusions. For brevity, we have omitted some proofs which can be found in [10].

Related Work. We summarize results on the emptiness problem, the value problem, and the isolation problem. The undecidability of the emptiness problem for probabilistic automata with non-extremal thresholds was shown for finite words in [14] and for infinite words in [8]. The emptiness problem for 2-HPAs (and higher) with non-extremal thresholds is also undecidable [4,8]. When the

cut-point is isolated, the decidability of the emptiness problem for general probabilistic automata is not known. However, for unary PFAs [6] and eventually weakly ergodic PFAs [13] the emptiness problem is decidable when the cut-point is isolated. Eventually weakly ergodic PFAs are incomparable to HPAs considered here; Fig. 1(c) in [13] is an example of a HPA that is not eventually weakly ergodic. The value problem is undecidable for PFAs [5], even for extremal thresholds [16]; it is known to be $\mathbf{\Pi_2^0}$-complete [13]. The problem of checking if the value is 1 was shown to be **PSPACE**-complete for *leak-tight automata* [15] which is sub-class of probabilistic automata that includes HPAs considered here. For value other than 1, no decidability results are known (other than those presented here). The isolation problem was shown to be $\mathbf{\Pi_2^0}$-complete [13] for general probabilistic automata.

2 Preliminaries

We assume that the reader is familiar with probability distributions, stochastic matrices finite-state automata, regular languages, Muller automata and ω-regular languages. The set of natural numbers will be denoted by $\mathbb{N}$, the closed unit interval by $[0, 1]$ and the open unit interval by $(0, 1)$. The power-set of a set X will be denoted by 2^X. The absolute value of a real number r shall be denoted by $|r|$. A non-negative rational number x is uniquely represented as a fraction $\frac{y}{z}$ where $y, z \in \mathbb{N}$ are relatively prime to each other, and $y \leq z$. In this case, we define the *size* of x to be the number of bits in the binary representation of z.

Sequences. Given a finite set S, $|S|$ denotes the cardinality of S. Given a sequence (finite or infinite) $\kappa = s_0 s_1 \dots$ over S, $|\kappa|$ will denote the length of the sequence (for infinite sequence $|\kappa|$ will be ω), and $\kappa[i]$ will denote the ith element s_i of the sequence with $\kappa[0]$ being the first. We will use ϵ to denote the (unique) empty string/sequence. For natural numbers i, j, $i \leq j < |\kappa|$, $\kappa[i : j]$ is the sequence $s_i \dots s_j$. For $i < |\kappa|$, $\kappa[i : \infty]$ is the sequence $s_i s_{i+1} \dots$ if $|\kappa| = \omega$, and is the sequence $s_i \dots s_{|\kappa|-1}$ if $|\kappa|$ is finite. As usual S^* will denote the set of all finite sequences/strings/words over S, S^+ will denote the set of all finite non-empty sequences/strings/words over S and S^ω will denote the set of all infinite sequences/strings/words over S. We will use u, v, w to range over elements of S^*, α, β, γ to range over infinite words over S^ω.

Given $u \in S^*$ and $\kappa \in S^* \cup S^\omega$, $u\kappa$ is the sequence obtained by concatenating the two sequences in order. Given $\mathsf{L}_1 \subseteq S^*$ and $\mathsf{L}_2 \subseteq S^* \cup S^\omega$, the set $\mathsf{L}_1\mathsf{L}_2$ is defined to be $\{u\kappa \mid u \in \mathsf{L}_1 \text{ and } \kappa \in \mathsf{L}_2\}$. Given $u \in S^+$, the word u^ω is the unique infinite sequence formed by repeating u infinitely often. For an infinite word $\alpha \in S^\omega$, we write $\mathsf{inf}(\alpha) = \{s \in S \mid s = \alpha[i] \text{ for infinitely many } i\}$.

Probabilistic Automaton (PA). Informally, a PA is like a finite-state deterministic automaton except that the transition function from a state on a given input is described as a probability distribution which determines the probability of the next state.

Definition 1. *A* finite-state probabilistic automaton (PA) *over a finite alphabet* Σ *is a tuple* $\mathcal{A} = (Q, q_s, \delta, \mathsf{Acc})$ *where* Q *is a finite set of states,* $q_s \in Q$ *is the initial state,* $\delta : Q \times \Sigma \times Q \to [0,1]$ *is the transition relation such that for all* $q \in Q$ *and* $a \in \Sigma$, $\delta(q,a,q')$ *is a rational number and* $\sum_{q' \in Q} \delta(q,a,q') = 1$, *and* Acc *is an* acceptance condition *(to be defined later).*

Notation: The transition function δ of PA $\mathcal{A}$ on input a can be seen as a square matrix δ_a of order $|Q|$ with the rows labeled by "current" state, columns labeled by "next state" and the entry $\delta_a(q,q')$ equal to $\delta(q,a,q')$. Given a word $u = a_0 a_1 \dots a_n \in \Sigma^+$, δ_u is the matrix product $\delta_{a_0}\delta_{a_1}\dots\delta_{a_n}$. For the empty word $\epsilon \in \Sigma^*$ we take δ_ϵ to be the identity matrix. Finally for any $Q_0 \subseteq Q$, we say that $\delta_u(q, Q_0) = \sum_{q' \in Q_0} \delta_u(q,q')$. Given a state $q \in Q$ and a word $u \in \Sigma^+$, $\mathsf{post}(q,u) = \{q' \mid \delta_u(q,q') > 0\}$. For a set $C \subseteq Q$, $\mathsf{post}(C,u) = \cup_{q \in C} \mathsf{post}(q,u)$.

Intuitively, the PA starts in the initial state q_s and if after reading $a_0, a_1 \dots, a_i$ it is in state q, then the PA moves to state q' with probability $\delta_{a_{i+1}}(q,q')$ on symbol a_{i+1}. A *run* of the PA $\mathcal{A}$ starting in a state $q \in Q$ on an input $\kappa \in \Sigma^* \cup \Sigma^\omega$ is a sequence $\rho \in Q^* \cup Q^\omega$ such that $|\rho| = 1 + |\kappa|, \rho[0] = q$ and for each $i \geq 0$, $\delta_{\kappa[i]}(\rho[i], \rho[i+1]) > 0$. Unless otherwise stated, a *run* for us will mean a run starting in the initial state q_s.

Given a word $\kappa \in \Sigma^* \cup \Sigma^\omega$, the PA $\mathcal{A}$ can be thought of as a (possibly infinite-state) (sub)-Markov chain. The set of states of this (sub)-Markov Chain is the set $\{(q,v) \mid q \in Q, v \text{ is a prefix of } \kappa\}$ and the probability of transitioning from (q,v) to (q',u) is $\delta_a(q,q')$ if $u = va$ for some $a \in \Sigma$ and 0 otherwise. This gives rise to the standard σ-algebra on Q^ω defined using cylinders and the standard probability measure on (sub)-Markov chains [17,21]. We shall henceforth denote the σ-algebra as $\mathcal{F}_{\mathcal{A},\kappa}$ and the probability measure as $\mu_{\mathcal{A},\kappa}$.

Acceptance Conditions and PA Languages. The language of a PA $\mathcal{A} = (Q, q_s, \delta, \mathsf{Acc})$ over an alphabet Σ is defined with respect to the acceptance condition Acc and a threshold $x \in [0,1]$. We consider two kinds of acceptance conditions.

Finite acceptance: When defining languages over finite words, the acceptance condition Acc is given in terms of a finite set $Q_f \subseteq Q$. In this case we call the PA $\mathcal{A}$, a probabilistic finite automaton (PFA). Given a finite acceptance condition $Q_f \subseteq Q$ and a finite word $u \in \Sigma^*$, a run ρ of $\mathcal{A}$ on u is said to be accepting if the last state of ρ is in Q_f. The set of accepting runs on $u \in \Sigma^*$ is measurable [21] and we denote its measure by $\mathsf{P}_\mathcal{A}(u)$. Note that $\mathsf{P}_\mathcal{A}(u) = \delta_u(q_s, Q_f)$. Given a PFA, a rational threshold $x \in [0,1]$ and the language of finite words $\mathsf{L}_{>x}(\mathcal{A}) = \{u \in \Sigma^* \mid \mathsf{P}_\mathcal{A}(u) > x\}$ is the set of finite words accepted by $\mathcal{A}$ with probability $> x$.

Muller acceptance: For Muller acceptance, the acceptance condition Acc is given in terms of a finite set $F \subseteq 2^Q$. In this case, we call the PA $\mathcal{A}$, a probabilistic Muller automaton (PMA). Given a Muller acceptance condition $F \subseteq 2^Q$, a run ρ of $\mathcal{A}$ on an infinite word $\alpha \in \mathcal{A}$ is said to be *accepting* if $\mathsf{inf}(\rho) \in F$. Once again,

the set of accepting runs is measurable [21]. Given a word α, the measure of the set of accepting runs is denoted by $\mathsf{P}_{\mathcal{A}}(\alpha)$. Given a PMA $\mathcal{A}$, a rational threshold $x \in [0,1]$, the language of infinite words $\mathsf{L}_{>x}(\mathcal{A}) = \{\alpha \in \Sigma^\omega \mid \mathsf{P}_{\mathcal{A}}(\alpha) > x\}$ is the set of infinite words accepted by PMA $\mathcal{A}$ with probability $> x$.

Changing the cutpoint. The following proposition allows us to change non-extremal cutpoints. It is proved for PFAs in [18]. The same construction also works for PMAs.

Proposition 1. *For any* PA $\mathcal{A}$*, rationals* $x, y \in (0,1)$*, there is a* PA $\mathcal{B}$ *constructible in linear time such that* $\mathsf{L}_{>x}(\mathcal{A}) = \mathsf{L}_{>y}(\mathcal{B})$.

The Value Problem(s). For a PA $\mathcal{A}$, let $\mathsf{value}(\mathcal{A})$ denote the least upper bound of the set $\{\mathsf{P}_{\mathcal{A}}(u) \mid u \in \Sigma^*\}$ when $\mathcal{A}$ is a PFA and of the set $\{\mathsf{P}_{\mathcal{A}}(\alpha) \mid \alpha \in \Sigma^\omega\}$ when $\mathcal{A}$ is a PMA. The *value computation problem* for a PA is the problem of computing $\mathsf{value}(\mathcal{A})$ for a given $\mathcal{A}$. The *value decision problem* is the problem of deciding for a given PA $\mathcal{A}$ and a rational $x \in [0,1]$ whether $\mathsf{value}(\mathcal{A}) = x$.

The Isolation Decision Problem. For a PA $\mathcal{A}$, a rational threshold $x \in [0,1]$ is said to be an *isolated cut-point* of $\mathcal{A}$ if there is an $\epsilon > 0$ such that for each word κ (where $\kappa \in \Sigma^*$ when $\mathcal{A}$ is a PFA and $\kappa \in \Sigma^\omega$ otherwise), we have that $|\mathsf{P}_{\mathcal{A}}(\kappa) - x| > \epsilon$. If such an ϵ exists then ϵ is said to be *a degree of isolation*. The *isolation decision problem* is the problem of deciding for a given PA $\mathcal{A}$ and a rational $x \in [0,1]$ whether x is an isolated cutpoint of $\mathcal{A}$.

We have the following relation between the isolated cutpoint decision problem and the value decision problem.

Proposition 2. *For each* PA $\mathcal{A} = (Q, q_s, \delta, \mathsf{Acc})$ *and* $x \in (0,1)$*, there is a constructible* PA $\mathcal{B}$ *such that* $\mathsf{value}(\mathcal{B}) = \frac{1}{4}$ *iff* x *is not a isolated cutpoint of* $\mathcal{A}$.

3 Hierarchical Probabilistic Automata

Intuitively, a hierarchical probabilistic automaton is a PA such that the set of its states can be stratified into totally-ordered levels. From a state q on each letter a, the machine can transit with non-zero probability to at most one state in the same level as q, and all other probabilistic successors belong to higher levels.

Definition 2. *For* $k \in \mathbb{N}$*, a* k-hierarchical probabilistic automaton *(*HPA*) is a probabilistic automaton* $\mathcal{A} = (Q, q_s, \delta, \mathsf{Acc})$ *over alphabet* Σ *such that* Q *can be partitioned into* $k+1$ *sets* $Q_0, Q_1, \ldots, Q_k$ *satisfying the following properties:*

- $q_s \in Q_0$*;*
- *for every* i*,* $0 \leq i \leq k$ *and every* $q \in Q_i$ *and* $a \in \Sigma$*,* $|\mathsf{post}(q,a) \cap Q_i| \leq 1$*; and,*
- *for every* i*,* $0 < i \leq k$*,* $q \in Q_i$ *and* $a \in \Sigma$*,* $\mathsf{post}(q,a) \cap Q_j = \emptyset$ *for every* $j < i$.

For any k-HPA $\mathcal{A}$*, as given above, for* $0 \leq i \leq k$*, we call the elements of* Q_i*,* level i *states of* $\mathcal{A}$*. We call a* HPA *a* HPFA*/*HPMA *if* Acc *is a finite acceptance/Muller acceptance condition respectively.*

Let us fix a k-HPA $\mathcal{A} = (Q, q_s, \delta, \mathsf{Acc})$ over alphabet Σ. Observe that given any state $q \in Q_0$ and any word $\kappa \in \Sigma^* \cup \Sigma^\omega$, $\mathcal{A}$ has at most one run ρ on α where all states in ρ belong to Q_0. We now present a couple of useful definitions. A set $W \subseteq Q$ is said to be a *witness* set if W has at most one level 0 state, i.e., $|W \cap Q_0| \leq 1$. Observe that for any word $u \in \Sigma^*$, $\mathsf{post}(q_s, u)$ is a witness set, i.e., $|\mathsf{post}(q_s, u) \cap Q_0| \leq 1$. We will say a word $\kappa \in \Sigma^* \cup \Sigma^\omega$ (depending on whether $\mathcal{A}$ is an automaton on finite or infinite words) is *definitely accepted* from witness set W iff for every $q \in W$ with $q \in Q_i$ (for $0 \leq i \leq k$) there is an accepting run ρ on κ starting from q such that for every j, $\rho[j] \in Q_i$ and $\delta_{\kappa[j]}(\rho[j], \rho[j+1]) = 1$. In other words, κ is definitely accepted from witness set W if and only if κ is accepted from every state q in W by a run where you stay in the same level as q, and all transitions in the run are taken with probability 1. Observe that the set of all words definitely accepted from a witness set W is regular. Furthermore, its emptiness can be checked in **PSPACE**.

Proposition 3. *For any* HPA $\mathcal{A}$ *and witness set* W*, the language* $\mathsf{L}_W = \{\kappa | \kappa$ *is definitely accepted by* $\mathcal{A}$ *from* $W\}$ *is regular. The emptiness of* L_W *can be checked in* **PSPACE***.*

That the emptiness of L_W can be checked in **PSPACE** follows from the observation that $\mathsf{L}_W = \cap_{q \in W} \mathsf{L}_{\{q\}}$ and $\mathsf{L}_\emptyset$ (as defined above) is the set of all strings.

Definition 3. *A witness set* W *is said to be* good *if the language* L_W *(defined in Proposition 3) is non-empty.*

Witness sets play an important role in the acceptance of strings. This is characterized by the following Proposition.

Proposition 4. *For a* HPA $\mathcal{A}$*, threshold* $x \in [0, 1]$*, and word* κ*,* $\kappa \in \mathsf{L}_{>x}(\mathcal{A})$ *if and only if there is a non-empty witness set* W*,* $u \in \Sigma^*$ *and* $\kappa' \in \Sigma^* \cup \Sigma^\omega$ *such that* $\kappa = u\kappa'$*,* κ' *is definitely accepted by* $\mathcal{A}$ *from* W*, and* $\delta_u(q_0, W) > x$*.*

Proposition 4 immediately implies the following.

Proposition 5. *For a* HPMA $\mathcal{A} = (Q, q_s, \delta, \mathsf{Acc})$ *let* $\mathcal{GW}$ *be the set of good non-empty witness sets of* $\mathcal{A}$*. For* $W \in \mathcal{GW}$*, let* $\mathcal{A}_W = (Q, q_s, \delta, W)$ *be the* PFA *that has* W *as the set of its final states. Then* $\mathsf{value}(\mathcal{A}) = \mathsf{max}_{W \in \mathcal{GW}}\, \mathsf{value}(\mathcal{A}_W)$*.*

3.1 Regularity of HPAs with Isolated Cut-points

Probabilistic automata, though finite state, are known to recognize non-regular languages, whether we consider automata on finite or infinite words [1,7,19]. One important result due to Rabin [19] is that $\mathsf{L}_{>x}(\mathcal{A})$ is regular for any PFA $\mathcal{A}$ if x is isolated for $\mathcal{A}$. We extend this observation to any HPMA.

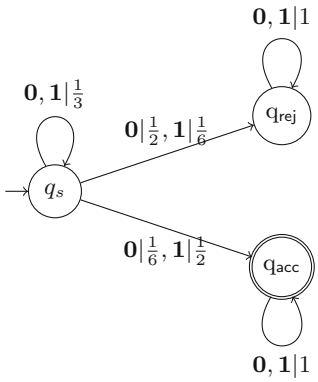


Fig. 1. $\mathcal{A}_{\mathsf{isolated}}$

Theorem 1. *Let $\mathcal{A}$ be a* HPMA *and let $x \in [0,1]$ be such that x is isolated for $\mathcal{A}$. Then $\mathsf{L}_{>x}(\mathcal{A})$ is ω-regular.*

Example 1. Consider the HPMA $\mathcal{A}_{\mathsf{isolated}}$ shown in Fig. 1. It has 3 states — $q_s, q_{\mathsf{rej}}, q_{\mathsf{acc}}$ — with q_s as initial state. The acceptance condition is given by $\{\{q_{\mathsf{acc}}\}\}$. For any n, let $x_n = \frac{3}{4}(1-(\frac{1}{3})^n)$. We will show a couple of properties about $\mathcal{A}_{\mathsf{isolated}}$ and x_n. First we show that x_n is an isolated cut-point with degree of isolation $\frac{1}{6}(\frac{1}{3})^n$, and second, that the set of infinite strings accepted with probability $> x_n$ is exactly $L_n = \mathbf{1}^n\{\mathbf{0},\mathbf{1}\}^\omega$.

For any $u \in \{\mathbf{0},\mathbf{1}\}^*$, let the acceptance and rejection probabilities of u be the probabilities of reaching the states $q_{\mathsf{acc}}, q_{\mathsf{rej}}$, respectively, on input u staring from q_s. Observe that every string in L_n, is accepted by $\mathcal{A}$, with probability greater than the acceptance probability of $u = 1^n0$, which is $\sum_{0\le i<n}(\frac{1}{3})^i\frac{1}{2} + (\frac{1}{3})^n\frac{1}{6}$ and is equal to $\frac{3}{4}(1-(\frac{1}{3})^n) + (\frac{1}{3})^n\frac{1}{6}$. Now, consider any input sequence not in L_n, i.e., sequence in $\{\mathbf{0},\mathbf{1}\}^\omega \setminus L_n$. The probability of rejecting any such string is $>$ the rejection probability the string $1^{n-1}01$. The rejection probability of $1^{n-1}01$ is $\sum_{0\le i<n-1}(\frac{1}{3})^i\frac{1}{6} + (\frac{1}{3})^{n-1}\frac{1}{2} + (\frac{1}{3})^n\frac{1}{6}$, which after some simplification, is $y = \frac{1}{4}(1-(\frac{1}{3})^{n-1}) + \frac{5}{3}(\frac{1}{3})^n$. From these observations, it follows that, for any $\kappa \notin L_n$, the probability of accepting κ is $< 1-y = \frac{3}{4}(1-(\frac{1}{3})^n) - \frac{1}{6}(\frac{1}{3})^n = x_n - \frac{1}{6}(\frac{1}{3})^n$. In addition, for any $\kappa \in L_n$, the probability of accepting κ is $> x_n + \frac{1}{6}(\frac{1}{3})^n$. Hence x_n is an isolated cut point with degree of isolation $\frac{1}{6}(\frac{1}{3})^n$.

4 Emptiness Under Isolation

We now show that the emptiness and universality problems are decidable for k-HPAs with isolated cut-points, even when the degree of isolation is not known. In order to establish the above result, we recall the definition of max-norms in matrices.

Definition 4. *For a $n \times n$ matrix δ, let δ_{ij} be the entry in i-th row and j-th column. We say that $\|\delta\| = \max_{i,j} |\delta_{ij}|$.*

For the rest of this section, we fix the input alphabet Σ. The decision procedure for checking emptiness and universality of k-HPAs depends on Lemma 1, which states that the "effect" of input word u on a k-HPFA $\mathcal{A}$ can be approximated by a short word v upto a given degree of approximation. The Lemma shows that for each ϵ, the matrix $\delta_u - \delta_v$ has max-norm $\leq \epsilon$.

Lemma 1. *Given a k-HPA $\mathcal{A} = (Q, q_s, \delta, \mathsf{Acc})$ and a rational $0 < \epsilon < 1$, there is a computable $\ell_{\mathcal{A},\epsilon} \in \mathbb{N}$ such that for each word $u \in \Sigma^*$, there is a word $v \in \Sigma^*$ such that $|v| \leq \ell_{\mathcal{A},\epsilon}$ and $\|\delta_u - \delta_v\| < \epsilon$. Furthermore $\ell_{\mathcal{A},\epsilon} \leq (\log\lceil \frac{2^{(b+1)k} n^{2k}}{\epsilon} \rceil)^k 2^{(b+2)k} n^{n+k}$ where $n = |Q|$ and b is the maximum size of the transition probabilities.*

We sketch the key ideas of the proof of Lemma 1. The proof proceeds by induction on k.

- We first observe that if $\mathcal{A}$ is a 0-HPA, then all transition probabilities are either 0 or 1. Hence the stochastic matrix δ_u is such that each entry is either 0 or 1 and each row consists of exactly one non-zero entry. Since there are only n^n matrices, if $|u| > n^n$ then there will be $i < j$ such that matrices $\delta_{u[0:i]}$ and $\delta_{u[0:j]}$ are the same. So, we can remove the word $u[i+1:j]$ from u without affecting the probability of transitioning from one state to another.
- Suppose that we have established the Lemma for $k = k_0$. In the induction step, we have to prove it for $k = k_0 + 1$. Fix a level 0 state q of the PA $\mathcal{A}$. For each prefix w of u, it is the case that there is at most one level 0 state in $\mathsf{post}(q, w)$. Assume that there is exactly one level 0 state in $\mathsf{post}(q, w)$. For each $i < |u|$, we will say that there is a leak at position i if on the input $u[i]$, some probability moves to higher levels. Now, between two consecutive leaks, the automaton $\mathcal{A}$ is essentially a k_0-HPA obtained by moving all states down one level. Thus, we can use the Induction Hypothesis to shorten the words between leaks. After we reach a point when there are too "many" leaks, the probability of being in level 0 is small and can be ignored. This informal argument only shows that the qth row of δ_u can be approximated by a short word. Some bookkeeping is needed to ensure that the same short word works for every row.

Using Lemma 1, we can show that for a k-HPA, $\mathsf{value}(\mathcal{A})$ can be computed within a given degree of accuracy.

Theorem 2. *There is an algorithm, which given a k-HPA $\mathcal{A} = (Q, q_s, \delta, \mathsf{Acc})$, and a rational $\epsilon \in (0, 1)$ computes x such that $|\mathsf{value}(\mathcal{A}) - x| \leq \epsilon$. The algorithm is exponential in $\mathsf{poly}(\log(\frac{1}{\epsilon}))$ and doubly exponential in the size of $\mathcal{A}$.*

Proof. The algorithm for the case when $\mathcal{A}$ is a HPFA works as follows. Given $\mathcal{A}$ and ϵ as given in the lemma, the algorithm computes $\ell_{\mathcal{A},\frac{\epsilon}{n}}$ where $n = |Q|$, enumerates all input sequence of length at most $\ell_{\mathcal{A},\frac{\epsilon}{n}}$, computes and outputs

the maximum of the acceptance probabilities of these strings. If x is the value output by the algorithm, using Lemma 1, it is easy to see that $|\mathsf{value}(\mathcal{A}) - x| \leq \epsilon$. The time bounds follow from the bound on $\ell_{\mathcal{A},\frac{\epsilon}{n}}$ in Lemma 1. For the case of HPMA, we appeal to Proposition 5 which allows us to approximate the value using HPFAs. □

The above algorithm to approximate the value of a HPA immediately gives us an algorithm that given a HPA $\mathcal{A}$ and a rational x such that x is an isolated cutpoint of $\mathcal{A}$ checks if the regular language $\mathsf{L}_{>x}(\mathcal{A})$ is empty or not, even if a degree of isolation is not known. The algorithm is obtained as follows. We progressively compute the value of $\mathcal{A}$ with increasing precision. Suppose at some point the value is approximated by v with precision ϵ. If $v - \epsilon > x$ then we know that $\mathcal{A}$ has a non-empty language. On the other hand, if $v + \epsilon < x$ then $\mathcal{A}$'s language is empty. Since x is isolated, we are guaranteed that eventually the precision ϵ is low enough to ensure that one of these two conditions hold. This is carried out in the following Theorem.

Input: Integer k, a k-HPA $\mathcal{A} = (Q, q_s, \delta, \mathsf{Acc})$ and rational $x \in [0, 1]$
Output: YES if $\mathsf{L}_{>x}(\mathcal{A}) = \emptyset$ and NO if $\mathsf{L}_{>x}(\mathcal{A}) \neq \emptyset$

$n \leftarrow |Q|$
approx_value $\leftarrow 0$
$\epsilon \leftarrow \frac{1}{2}$
if $\mathcal{A}$ *is a* HPFA **then**
  $\mathcal{GW} \leftarrow \{\mathsf{Acc}\}$
else
  $\mathcal{GW} \leftarrow \{W \mid W \subseteq Q, W \neq \emptyset, W$ is a good witness$\}$
end
while true **do**
  Compute $\ell_{\mathcal{A},\frac{\epsilon}{n}}$ as given in Lemma 1
  approx_value $\leftarrow \max_{W \in \mathcal{GW}, v \in \Sigma^*, |v| \leq \ell_{\mathcal{A},\frac{\epsilon}{n}}} \delta_v(q_s, W)$
  if approx_value $> x$ **then**
    return NO
  else
    if approx_value $< x - \epsilon$ **then**
      return YES
    else
      $\epsilon \leftarrow \frac{\epsilon}{2}$
    end
  end
end

Fig. 2. Procedure for checking emptiness of HPAs

Theorem 3. *The algorithm in Fig. 2 solves the following problem: Given a k-*HPA $\mathcal{A} = (Q, q_s, \delta, \mathsf{Acc})$ *and a rational $x \in (0,1)$ such that x is an isolated cut-point for $\mathcal{A}$, decide if $\mathsf{L}_{>x}(\mathcal{A})$ is empty.*

Proof. Let the number of states of $\mathcal{A}$ be n. Let ϵ_m be the value of variable ϵ at the beginning of the mth iteration of the while loop. Clearly $\epsilon_m = \frac{1}{2^m}$. Consider first the case when $\mathcal{A}$ is a HPFA. The case when $\mathcal{A}$ is a HPMA follows a similar argument.

Clearly if the procedure outputs NO then $\mathsf{L}_{>x}(\mathcal{A}) \neq \emptyset$. Now suppose that the the algorithm outputs YES. Let ϵ_{m_0} be the value of ϵ when the algorithm outputs YES. As the program outputs YES, for each word w such that $|w| \leq \ell_{\mathcal{A},\frac{\epsilon_{m_0}}{n}}$, we have that $\delta_w(q_s, Q_f) + \epsilon_{m_0} < x$. Fix a finite word u. Thanks to Lemma 1, there is finite word v such that $|v| \leq \ell_{\mathcal{A},\frac{\epsilon_{m_0}}{n}}$ and $\delta_u(q_s, Q_f) < \delta_v(q_s, Q_f) + \epsilon_{m_0} < x$. Thus, if the algorithm outputs YES then $\mathsf{L}_{>x}(\mathcal{A}) = \emptyset$. Notice that if the algorithm terminates then it gives the correct answer even if x is not isolated.

We claim that the algorithm in Fig. 2 terminates if $\mathsf{L}_{>x}(\mathcal{A}) \neq \emptyset$ or if $\mathsf{value}(\mathcal{A}) < x$. If $\mathsf{L}_{>x}(\mathcal{A}) \neq \emptyset$ then fix a word u such that $\delta_u(q_s, Q_f) > x$. Let $\epsilon' = \delta_u(q_s, Q_f) - x$. Let m_0 be the smallest integer such that $n\epsilon_{m_0} < \epsilon'$. Thanks to Lemma 1, there is a finite word v such that $|v| \leq \ell_{\mathcal{A},\frac{\epsilon_{m_0}}{n}}$ and $\delta_v(q_s, Q_f) > \delta_u(q_s, Q_f) - n\epsilon_{m_0} = x + \epsilon' - n\epsilon_{m_0} > x$. Thus $\mathsf{approx\_value} > x$ in the m_0th unrolling of the while loop and the algorithm terminates.

If $\mathsf{value}(\mathcal{A}) < x$ then let $\epsilon' = x - \mathsf{value}(\mathcal{A})$. Let m_0 be the smallest integer such that $\epsilon_{m_0} < \epsilon'$. It is easy to see that the algorithm will terminate in the m_0th unrolling of the loop as for every word w, it is the case that $\delta_w(q_s, Q_f) + \epsilon_{m_0} \leq (x - \epsilon') + \epsilon_{m_0} < x$.

The Theorem follows from the fact that if x is an isolated cutpoint of $\mathcal{A}$ and $\mathsf{L}_{>x}(\mathcal{A}) = \emptyset$ then $\mathsf{value}(\mathcal{A}) < x$. □

Next, we show that if x is isolated for a PA $\mathcal{A}$ then we can decide if $\mathsf{L}_{>x}(\mathcal{A})$ is contained in/contains a given regular language R (where R is a regular language over finite or infinite words depending on whether $\mathcal{A}$ is a HPFA or a HPMA). Observe that this also implies that the problem of checking whether $\mathsf{L}_{>x}(\mathcal{A})$ is universal or not is also decidable if x is an isolated cutpoint of $\mathcal{A}$.

Theorem 4. *Let $\bowtie \in \{\subseteq, \supseteq, =\}$. There is an algorithm that given a regular language R, a k-*HPA $\mathcal{A} = (Q, q_s, \delta, \mathsf{Acc})$*, a rational $x \in (0,1)$ such that x is an isolated cut-point for $\mathcal{A}$, decides if $\mathsf{L}_{>x}(\mathcal{A}) \bowtie R$.*

5 On the Value Decision Problem

For a PFA $\mathcal{A}$, the problem of checking if $x \in [0,1]$ is an isolated cut-point is $\mathbf{\Sigma_2^0}$-complete [13]. Observe that 1 is an isolated cutpoint of a PFA $\mathcal{A}$ iff $\mathsf{value}(\mathcal{A}) < 1$. An immediate consequence is that the value decision problem for PFAs is $\mathbf{\Pi_2^0}$-complete. For HPFAs, the problem of checking if 1 is isolated is known to be **PSPACE**-complete [15]. The same result holds for checking if 0 is an isolated

cutpoint for a HPA. We now show that the problem checking whether $x \in (0,1)$ is an isolated cutpoint for a HPA is **R.E.**-complete and the value problem is **co-R.E.**-complete. Hence, the isolated cut point decision problem and the value decision problem are simpler for HPAs. We start by proving that the value problem is **co-R.E.**-complete. The proof of containment in **co-R.E.** relies on Lemma 1, which allows us to approximate the effect of each finite word on an automaton by a short word. The hardness result is obtained by a modification of the proof of undecidability of emptiness problem for the 2-HPAs [2–4].

Theorem 5. *For each $k \geq 2$, the value decision problems for k-*HPFA*s and for k-*HPMA*s are* **co-R.E.**-*complete.*

Proof. We first establish that the value problem is in **co-R.E.** Consider first the case for HPFAs. Let $\mathcal{A} = (Q, q_s, \delta, \mathsf{Acc})$ be a HPFA. Let $\mathsf{isValue}(\mathcal{A}, x)$ be the predicate

$$\mathsf{isValue}(\mathcal{A}, x) = (\forall v \in \Sigma^*.\ \mathsf{P}_{\mathcal{A}}(v) \leq x) \wedge \forall m \in \mathbb{N}.\ (\exists u \in \Sigma^*.\ \mathsf{P}_{\mathcal{A}}(u) > x - \frac{1}{m}).$$

It is easy to see that $\mathsf{value}(\mathcal{A}) = x$ iff $\mathsf{isValue}(\mathcal{A}, x)$ is true.

Let $|Q| = n$ and let $\mathsf{isValueSm}(\mathcal{A}, x)$ be the predicate

$$\mathsf{isValueSm}(\mathcal{A}, x) = (\forall v \in \Sigma^*.\ \mathsf{P}_{\mathcal{A}}(v) \leq x) \wedge \\ \forall m \in \mathbb{N}.\ (\exists v \in \Sigma^*.\ |v| \leq \ell_{\mathcal{A}, \frac{1}{mn}} \wedge \mathsf{P}_{\mathcal{A}}(v) > x - \tfrac{1}{2m}).$$

It is easy to see that if $\mathsf{isValueSm}(\mathcal{A}, x)$ is true then so is $\mathsf{isValue}(\mathcal{A}, x)$.

Assume now that $\mathsf{isValue}(\mathcal{A}, x)$ is true. Then for each $m \in \mathbb{N}$, there is a $u \in \Sigma^*$ such that $\mathsf{P}_{\mathcal{A}}(u) > x - \frac{1}{m}$. Fix m, u. Thanks to Lemma 1 there is a v such that $|v| \leq \ell_{\mathcal{A}, \frac{1}{mn}}$ and

$$\delta_u(q_s, q) - \frac{1}{mn} < \delta_v(q_s, q) < \delta_u(q_s, q) + \frac{1}{mn} \text{ for each } q \in Q. \tag{1}$$

Fix v. Therefore we get from Eq. 1 above that

$$\delta_v(q_s, Q_f) > \sum_{q \in Q_f} (\delta_u(q_s, q) - \frac{1}{mn}) = \delta_u(q_s, Q_f) - \frac{|Q_f|}{mn} > x - \frac{1}{m} - \frac{1}{m}.$$

It follows that $\mathsf{isValueSm}(\mathcal{A}, x)$ is also true if $\mathsf{isValue}(\mathcal{A}, x)$ is true. Hence, $\mathsf{value}(\mathcal{A}) = x$ iff $\mathsf{isValueSm}(\mathcal{A}, x)$ is true. Note that the problem of checking that for given v, if $\mathsf{P}_{\mathcal{A}}(v) \leq x$ is decidable. Also the problem of checking that given $m \in \mathbb{N}$, $(\exists v \in \Sigma^*.\ |v| \leq \ell_{\mathcal{A}, \frac{1}{mn}} \wedge \mathsf{P}_{\mathcal{A}}(v) > x - \frac{1}{2m})$ is decidable since $\ell_{\mathcal{A}, \frac{1}{mn}}$ is computable. Thus, the value problem is in **co-R.E.**

Now consider the theorem for HPMAs. Let $\mathcal{A} = (Q, q_s, \delta, \mathsf{Acc})$ be a HPMA. Let $\mathcal{GW}$ be the set of good non-empty witness sets of $\mathcal{A}$. For $W \in \mathcal{GW}$, let $\mathcal{A}_W = (Q, q_s, \delta, W)$ be the PFA that has W as the set of its final states. Thanks to Proposition 5, we have that $\mathsf{value}(\mathcal{A}) = \mathsf{max}_{W \in \mathcal{GW}}\ \mathsf{value}(\mathcal{A}_W)$. This implies that $\mathsf{value}(\mathcal{A}) = x$ iff one of the predicates $\{\mathsf{isValue}(\mathcal{A}_W, x) \mid W \in \mathcal{GW}\}$ is true. The upper bound follows in this case.

The lower bound is proved by a modification of the proof of undecidability of emptiness problem for the 2-HPAs [2–4]. □

Using Proposition 2, we can convert the non-isolation decision problem to the value decision problem. Thus the problem of checking whether a cut-point x is isolated for a HPA $\mathcal{A}$ is in **R.E.**. We can show that the non-isolation decision problem is **co-R.E.**-hard also using the same reduction that is used to prove **co-R.E.**-hardness of the value problem. This yields the following Theorem.

Theorem 6. *For each $k \geq 2$, the isolation decision problems for k-*HPFA*s and k-*HPMA*s are* **R.E.***-complete.*

5.1 Computing the Value of 1-HPAs

In this section, we give an **EXPTIME** algorithm for computing the value of a 1-HPA. The key technical observation to make this possible is a necessary and sufficient condition for when x is the value of a 1-HPFA. The observation is that there is always an exponentially bounded "ultimately periodic" witness for the value being x; this is the content of the next Lemma.

Lemma 2. *Let $\mathcal{A} = (Q, q_s, \delta, Q_f)$ be an* 1-HPFA *over an alphabet Σ, and $n = |Q|$. Then, for any x, $x = \mathsf{value}(\mathcal{A})$ iff there is no string that is accepted by $\mathcal{A}$ with probability $> x$ and at least one of the following conditions is satisfied.*

1. *$\exists u \in \Sigma^*$ such that $|u| \leq 2^n$ and $\mathsf{P}_{\mathcal{A}}(u) = x$.*
2. *$\exists u, v \in \Sigma^*$ such that $|u|, |v| \leq 2^n$, there exists a good witness set $W \subseteq Q_1$ such that $W \subseteq \mathsf{post}(q_s, u)$, $\mathsf{post}(W, v) \subseteq W$, $\mathsf{post}(q_s, u) \cap Q_0 = \mathsf{post}(q_s, uv) \cap Q_0$, $\forall i \geq 0$, $\delta_{uv^{i+1}}(q_s, W) > \delta_{uv^i}(q_s, W)$ and $\lim_{i \to \infty} \delta_{uv^i}(q_s, W) = x$.*

Condition 1 of the lemma corresponds to the case when there is an input string that is accepted with the maximum possible probability $\mathsf{value}(\mathcal{A})$. If there is no input string that is accepted with probability $\mathsf{value}(\mathcal{A})$, then Condition 2 of the lemma asserts that there are finite sequences u, v and a good witness set W, such that $\delta_{uv^i}(q_s, W)$ increases monotonically with increasing values of i, and the limit of this monotonic sequence equals $\mathsf{value}(\mathcal{A})$.

The next observation bounds the size of the probability of reaching a set of states C on an input u, as a function of $|u|$.

Lemma 3. *Let $\mathcal{A} = (Q, q_s, \delta, \mathsf{Acc})$ be a* 1-HPA *over an alphabet Σ and $n = |Q|$. Then, for any $u \in \Sigma^+$, $q \in Q_0$ and $C \subseteq Q$, the size of $\delta_u(q, C)$ is $\leq |u| n r$ where r is the maximum of the sizes of the transition probabilities of $\mathcal{A}$.*

Proof. The lemma is proved by a simple induction on $|u|$. In the base case, $|u| = 1$, the observation follows from the fact that $\delta_u(q, C)$ is the sum of atmost n transition probabilities of $\mathcal{A}$. For the inductive step, assume that the observation is true for all strings of length $\leq k$. Let $u = av$ be a string of length $k+1$ where $a \in \Sigma$ and $v \in \Sigma^k$. Clearly, $\delta_u(q, C) = \sum_{q' \in Q_1} \delta_a(q, q') \delta_v(q', C) + \delta_a(q, q_1) \delta_v(q_1, C)$ where $q_1 \in Q_0$ be such that $\delta_a(q, q_1) > 0$. Observe that, for $q' \in Q_1$, $\delta_v(q', C)$ is either 0 or 1. Now, it is easy to see that size of $\delta_u(q, C)$ is $\leq$ the sum of the sizes of $\delta_a(q, q')$ for less than n distinct $q' \in Q_1$, the sizes of $\delta_a(q, q_1)$ and $\delta_v(q_1, C)$. Using the induction hypothesis for v and observing that the sizes of $\delta_a(q, q')$, $\delta_a(q, q_1)$ are both $\leq r$, we get the desired result. ◻

The last technical lemma we need, bounds the size of the value of a 1-HPFA using Lemmas 2 and 3.

Lemma 4. *Let $\mathcal{A} = (Q, q_s, \delta, Q_f)$ be a* 1-HPFA *over an alphabet Σ and $n = |Q|$. Then, the size of* $\mathsf{value}(\mathcal{A})$ *is $\leq 4rn2^n$ where r is the maximum of the sizes of the transition probabilities of $\mathcal{A}$.*

Proof. Let $x = \mathsf{value}(\mathcal{A})$. Clearly no string is accepted by $\mathcal{A}$ with probability $> x$. Further, from Lemma 2, we see that one of the conditions (1) or (2), stated there, is satisfied. Suppose condition (1) is satisfied. Then, there is a string $u \in \Sigma^*$, such that $|u| \leq 2^n$ and $x = \mathsf{P}_{\mathcal{A}}(u)$. Now, our result follows from Lemma 3.

Now, suppose condition (2) of Lemma 2 is satisfied; let u, v, W be as specified in that condition. Let $\mathsf{post}(q_s, u) \cap Q_0 = \mathsf{post}(q_s, uv) \cap Q_0 = \{q_1\}$. It is easy to see that

$$\begin{aligned}\lim_{i\to\infty} \delta_{uv^i}(q_s, W) &= \delta_u(q_s, W) + \delta_u(q_s, q_1)\delta_v(q_1, W) \textstyle\sum_{i=0}^{\infty} (\delta_v(q_1, q_1))^i \\ &= \delta_u(q_s, W) + \delta_u(q_s, q_1) \tfrac{\delta_v(q_1, W)}{1-\delta_v(q_1, q_1)}\end{aligned}$$

Since, $|u|, |v| \leq 2^n$, we see from Lemma 3 that the sizes of $\delta_u(q_s, W), \delta_v(q_1, W)$, $\delta_u(q_s, q_1)$ and $\delta_v(q_1, q_1)$ are all $\leq rn2^n$. From this and the above equation, it is easy to see that the size of $\lim_{i\to\infty} \delta_{uv^i}(q_s, W)$ is atmost the sum of the sizes of $\delta_u(q_s, W)$, $\delta_v(q_1, W)$, $\delta_u(q_s, q_1)$, and $\delta_v(q_1, q_1)$. From this we observe that the size of $\lim_{i\to\infty} \delta_{uv^i}(q_s, W)$, and hence the size of x, is $\leq 4rn2^n$. □

We are now ready to present the main result of this section — an exponential time algorithm to compute the value of a 1-HPA.

Theorem 7. *The value of a* 1-HPA $\mathcal{A}$ *can be computed in exponential time. The value decision problems for* 1-HPA*s and* 1-HPMA*s are in* **EXPTIME** *and are* **PSPACE***-hard.*

Proof. First we consider the case for HPFAs. Let $\mathcal{A} = (Q, q_s, \delta, \mathsf{Acc})$ be the given HPFA over an alphabet Σ, and $n = |Q|$. There is a näive double exponential time algorithm that computes $\mathsf{value}(\mathcal{A})$ using Lemma 2. Such an algorithm enumerates all triples (u, v, W) such that $|u|, |v| \leq 2^n$, $W \subseteq Q_1$ and all the properties stated in condition (2) of Lemma 2 are satisfied. It computes $\mathsf{value}(\mathcal{A})$ to be the maximum of $\lim_{i\to\infty} \delta_{uv^i}(q_s, W)$ over all such triples (u, v, W). It is easy to see that such an algorithm is of time complexity doubly exponential in n.

Now, we give an algorithm, that computes $\mathsf{value}(\mathcal{A})$, of time complexity only singly exponential in n. Let $N = 4rn2^n$ and $M = 2^N$ where r is the maximum of the sizes of the transition probabilities of $\mathcal{A}$. From Lemma 4, we see that the size of $\mathsf{value}(\mathcal{A})$ is $\leq N$. Let $\mathsf{value}(\mathcal{A}) = \frac{y}{z}$. The above observation implies that $y, z \leq M$. Now, we employ an approach based on binary search on rationals [20] to compute the exact value of $\mathsf{value}(\mathcal{A})$. Essentially, this approach divides the unit interval $[0, 1]$ into $2M^2$ sub-intervals of equal length, i.e., each of length $\frac{1}{2M^2}$. Then, using binary search that employs queries of the form "$\mathsf{L}_{>x}(\mathcal{A}) = \emptyset$?", where $x = \frac{k}{2M^2}$ for some $k \leq 2M^2$, this approach determines the unique integer

$\ell \leq 2M^2$ such that $\mathsf{value}(\mathcal{A})$ is in the interval $[\frac{\ell}{2M^2}, \frac{\ell+1}{2M^2})$. (Every such interval has atmost one rational number of the form $\frac{y_1}{z_1}$ where $y_1, z_1 \leq M$).

Once such an interval is identified, the exact value of $\mathsf{value}(\mathcal{A})$ is computed using a simple algorithm, given in [20], of complexity $O(\log M)$, i.e., of complexity $O(N)$. Each query of the form "$\mathsf{L}_{>x}(\mathcal{A}) = \emptyset$?" can be answered using the algorithm for the emptiness problem for 1-HPA as given in [3,11]; the algorithm given in [11] is of complexity linear in the size of x and exponential in n. Since the size of x used in the above algorithm is $\leq N$, the time complexity of a single invocation of this algorithm during the binary search is seen to be $O(r8^n)$. Further more, there are at most N such invocations and hence the over all complexity of performing the binary search is $O(r^2 16^n)$. Furthermore, the complexity of the second step of the algorithm, i.e., the step in which the actual values of $\mathsf{value}(\mathcal{A})$ is computed, is also of time complexity $O(N)$. Hence the overall time complexity of the above algorithm for computing $\mathsf{value}(\mathcal{A})$ is $O(r^2 16^n)$.

For HPMAs, we use Proposition 5. Let $\mathcal{A}$ be a HPMA. $\mathcal{GW}$ be the set of good non-empty witness sets of $\mathcal{A}$. Using this proposition, we compute $\mathsf{value}(\mathcal{A})$ to be $\mathsf{max}_{W \in \mathcal{GW}}\, \mathsf{value}(\mathcal{A}_W)$, where $\mathcal{A}_W = (Q, q_s, \delta, W)$. Since $\mathsf{value}(\mathcal{A}_W)$ can be computed in time $O(r^2 16^n)$ and $|\mathcal{GW}| \leq 2^n$, we see that the time complexity of computing $\mathsf{value}(\mathcal{A})$ is $O(r^2 32^n)$.

Thus, it is easy to see that the value decision problem is in **EXPTIME**. It can be shown to be **PSPACE**-hard using the same techniques used to prove that the emptiness problem for 1-HPAs is **PSPACE**-hard in [3,11]. □

6 Conclusions

In this paper, we presented a number of results on HPAs. First, we showed that for a k-HPA, the effect of any string (i.e., the transition probability matrix of the string) can be approximated by that of a short string of bounded length, for a given precision. This can be used to approximate the value of a k-HPA with arbitrary precision, and decide the emptiness of the language of a k-HPA with an isolated cut-point. These observations allowed us to prove that the problem of computing the value of a k-HPA (for $k \geq 2$) is **co-R.E.**-complete. For a 1-HPA, we showed that it's value can be computed exactly in exponential time. A couple of problems for 1-HPAs remain open — the decidability of the isolation problem and the exact complexity of the value problem which has been shown to be in **EXPTIME**.

Acknowledgements. We thanks the anonymous reviewers for their useful comments and suggestions. Rohit Chadha was partially supported by NSF CNS 1314338 and NSF CNS 1553548. A. Prasad Sistla was partially supported by NSF CCF 1319754 and NSF CNS 1314485. Mahesh Viswanathan was partially supported by NSF CNS 1314485.

References

1. Baier, C., Größer, M.: Recognizing ω-regular languages with probabilistic automata. In: 20th IEEE Symposium on Logic in Computer Science, pp. 137–146 (2005)
2. Ben, Y.: Model Checking Open Probabilistic Systems using Hierarchical probabilistic automata. Ph.D. thesis, University of Illinois, Chicago (2016)
3. Ben, Y., Chadha, R., Sistla, A.P., Viswanathan, M.: Decidable and expressive classes of probabilistic automata. https://www.cs.uic.edu/pub/Sistla/Publications/HPAjournal2016.pdf (2016). Manuscript under review
4. Ben, Y., Sistla, A.P.: Model checking failure-prone open systems using probabilistic automata. In: Finkbeiner, B., Pu, G., Zhang, L. (eds.) ATVA 2015. LNCS, vol. 9364, pp. 148–165. Springer, Cham (2015). doi:10.1007/978-3-319-24953-7_11
5. Bertoni, A.: The solution of problems relative to probabilistic automata in the frame of the formal languages theory. In: Siefkes, D. (ed.) GI 1974. LNCS, vol. 26, pp. 107–112. Springer, Heidelberg (1975). doi:10.1007/3-540-07141-5_213
6. Chadha, R., Kini, D., Viswanathan, M.: Decidable problems for unary PFAs. In: Norman, G., Sanders, W. (eds.) QEST 2014. LNCS, vol. 8657, pp. 329–344. Springer, Cham (2014). doi:10.1007/978-3-319-10696-0_26
7. Chadha, R., Sistla, A.P., Viswanathan, M.: On the expressiveness and complexity of randomization in finite state monitors. J. ACM 56(5) (2009)
8. Chadha, R., Sistla, A.P., Viswanathan, M.: Probabilistic Büchi automata with non-extremal acceptance thresholds. In: International Conference on Verification, Model Checking and Abstract Interpretation, pp. 103–117 (2010)
9. Chadha, R., Sistla, A.P., Viswanathan, M.: Power of randomization in automata on infinite strings. Log. Methods Comput. Sci. **7**(3), 1–22 (2011)
10. Chadha, R., Sistla, A.P., Viswanathan, M.: Emptiness under isolation and the value problem for hierarchical probabilistic automata (2017). https://www.cs.uic.edu/pub/Sistla/Publications/ValueProblem.pdf
11. Chadha, R., Sistla, A.P., Viswanathan, M., Ben, Y.: Decidable and expressive classes of probabilistic automata. In: Pitts, A. (ed.) FoSSaCS 2015. LNCS, vol. 9034, pp. 200–214. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46678-0_13
12. Chadha, R., Sistla, A.P., Viswanathan, M.: Power of randomization in automata on infinite strings. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 229–243. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04081-8_16
13. Chadha, R., Sistla, A.P., Viswanathan, M.: Probabilistic automata with isolated cut-points. In: Chatterjee, K., Sgall, J. (eds.) MFCS 2013. LNCS, vol. 8087, pp. 254–265. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40313-2_24
14. Condon, A., Lipton, R.J.: On the complexity of space bounded interactive proofs (extended abstract). In: Symposium on Foundations of Computer Science, pp. 462–467 (1989)
15. Fijalkow, N., Gimbert, H., Oualhadj, Y.: Deciding the value 1 problem for probabilistic leaktight automata. In: IEEE Symposium on Logic in Computer Science, pp. 295–304 (2012)
16. Gimbert, H., Oualhadj, Y.: Probabilistic automata on finite words: Decidable and undecidable problems. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6199, pp. 527–538. Springer, Heidelberg (2010). doi:10.1007/978-3-642-14162-1_44

17. Kemeny, J.G., Snell, J.L.: Denumerable Markov Chains. Springer, New York (1976)
18. Paz, A.: Introduction to Probabilistic Automata. Academic Press, Orlando (1971)
19. Rabin, M.O.: Probabilistic automata. Inf. Control **6**(3), 230–245 (1963)
20. Kwek, S., Mehlhorn, K.: Optimal search for rationals. Inf. Process. Lett. **86**(1), 23–26 (2003)
21. Vardi, M.: Automatic verification of probabilistic concurrent finite-state programs. In: Symposium on Foundations of Computer Science, pp. 327–338 (1985)

Partial Derivatives for Context-Free Languages
From μ-Regular Expressions to Pushdown Automata

Peter Thiemann(✉)

University of Freiburg, Freiburg, Germany
thiemann@informatik.uni-freiburg.de

Abstract. We extend Antimirov's partial derivatives from regular expressions to μ-regular expressions that describe context-free languages. We prove the correctness of partial derivatives as well as the finiteness of the set of iterated partial derivatives. The latter are used as pushdown symbols in our construction of a nondeterministic pushdown automaton, which generalizes Antimirov's NFA construction.

Keywords: Automata and logic · Regular languages · Context-free languages · Pushdown automata · Derivatives

1 Introduction

Brzozowski derivatives [5] and Antimirov's partial derivatives [4] are well-known tools to transform regular expressions to finite automata and to define algorithms for equivalence and containment of regular languages [3,10]. Both automata constructions rely on the finiteness of the set of iterated derivatives. Brzozowski derivatives need to be considered up to similarity (commutativity, associativity, and idempotence for union) to obtain finiteness. Derivatives had quite some impact on the study of algorithms for regular languages on finite words and trees [6,15].

There are many studies of derivative structures for enhancements of regular expressions. While Brzozowski's original work covered extended regular expressions, partial derivatives were originally limited to simple expressions without intersection and complement. It is a significant effort to define partial derivatives for extended regular expressions [6]. Many further operators have been considered, among them shuffle operators [16], multi-tilde-bar expressions [7], expressions with multiplicities [12], approximate regular expressions [9], and many more. There have been a number of approaches to develop general frameworks for derivation: Caron and coworkers [8] abstract over the support for creating derivations, Thiemann [17] develops criteria for derivable language operators.

Recently, there has been practical interest in the study of derivatives and partial derivatives. Owens and coworkers [14] report a functional implementation with some extensions (e.g., character classes) to handle large character sets, which is partially rediscovering work on the FIRE library [18]. Might and coworkers [1,13] push beyond regular languages by implementing parsing for context-free languages using derivatives and demonstrate its feasibility in practice.

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 248–264, 2017.
DOI: 10.1007/978-3-662-54458-7_15

Winter and coworkers [19] study context-free languages in a coalgebraic setting. They use a notion of derivative to give three equivalent characterizations of context-free languages by grammars in weak Greibach normal form, behavioral differential equations, and guarded μ-regular expressions.

In this work, we focus on using derivatives for parsing of context-free languages. While Might and coworkers explore algorithmic issues, we investigate the correctness of context-free parsing with derivatives. To this end, we develop the theory of derivatives for μ-regular expressions, which extend regular expressions with a least fixed point operator. Our results are relevant for context-free parsing because μ-regular expressions are equivalent to context-free grammars in generating power. Compared to the work of Winter and coworkers [19], we do not require recursion to be guarded (i.e., we admit left recursion) and we focus on establishing the connection to pushdown automata. Unguarded recursion forces us to consider derivation by ε, which corresponds to an unfolding of a left-recursive μ-expression. Guarded expressions always admit a proper derivation by a symbol.

Our theory is the proper generalization of Antimirov's theory of partial derivatives to μ-regular expressions: our derivative function corresponds *exactly* to the transition function of the nondeterministic pushdown automaton that recognizes the same language. The pendant of Antimirov's finiteness result yields the finiteness of the set of pushdown symbols of this automaton.

2 Preliminaries

We write $\mathbb{N}$ for the set of natural numbers, $\mathbb{B} = \{\mathbf{ff}, \mathbf{tt}\}$ for the set of booleans, and $X \uplus Y$ for the disjoint union of sets X and Y. We consider total maps $m : X \to Y$ as sets of pairs in the usual way, so that $m \subseteq X \times Y$ and $\emptyset$ denotes the empty mapping. For $x_0 \in X$ and $y_0 \in Y$, the *map update* of m is defined as $m[y_0/x_0](x) = y_0$ if $x = x_0$ and $m[y_0/x_0](x) = m(x)$ if $x \neq x_0$.

For conciseness, we fix a finite set of symbols, Σ, as the underlying *alphabet*. We write Σ^* for the set of finite words over Σ, $\varepsilon \in \Sigma^*$ stands for the empty word, and $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. For $u, v \in \Sigma^*$, we write $|u| \in \mathbb{N}$ for the length of u and $u \cdot v$ (or just uv) for the concatenation of words.

Given languages $U, V, W \subseteq \Sigma^*$, concatenation extends to languages as usual: $U \cdot V = \{u \cdot v \mid u \in U, v \in V\}$. The Kleene closure is defined as the smallest set $U^* \subseteq \Sigma^*$ such that $U^* = \{\varepsilon\} \cup U \cdot U^*$. We write the *left quotient* as $U \backslash W = \{v \mid v \in \Sigma^*, \exists u \in U : uv \in W\}$. For a singleton language $U = \{u\}$, we write $u \backslash W$ for the left quotient.

Definition 1. *A (nondeterministic) finite automaton (NFA) is a tuple* $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ *where* Q *is a finite set of states,* Σ *an alphabet,* $\delta \subseteq Q \times \Sigma \times Q$ *the transition relation,* $q_0 \in Q$ *the initial state, and* $F \subseteq Q$ *the set of final states.*

Let $n \in \mathbb{N}$. *A* run *of* $\mathcal{A}$ *on* $w = a_0 \dots a_{n-1} \in \Sigma^*$ *is a sequence* $q_0 \dots q_n \in Q^*$ *such that, for all* $0 \leq i < n$, $(q_i, a_i, q_{i+1}) \in \delta$. *The run is* accepting *if* $q_n \in F$. *The language recognized by* $\mathcal{A}$ *is* $\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \exists\, \text{accepting run of } \mathcal{A} \text{ on } w\}$.

Definition 2. *A (nondeterministic) pushdown automaton (PDA) is a tuple* $\mathcal{P} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$ *where* Q *is a finite set of states,* Σ *the input alphabet,* Γ *the pushdown alphabet (a finite set),* $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \times Q \times \Gamma^*$ *is the transition relation,* $q_0 \in Q$ *is the initial state,* $Z_0 \in \Gamma$ *is the bottom symbol.*

A configuration of $\mathcal{P}$ *is a tuple* $c \in Q \times \Sigma^* \times \Gamma^*$ *of the current state, the rest of the input, and the current contents of the pushdown.*

The transition relation δ *gives rise to a binary stepping relation* $\vdash$ *on configurations defined by (for all* $q, q' \in Q$*,* $\alpha \in \Sigma \cup \{\varepsilon\}$*,* $Z \in \Gamma$*,* $\gamma, \gamma' \in \Gamma^*$*,* $v \in \Sigma^*$*):*

$$\frac{(q, \alpha, Z, q', \gamma') \in \delta}{(q, \alpha v, Z\gamma) \vdash (q', v, \gamma'\gamma)}$$

The language of the PDA is $\mathcal{L}(\mathcal{P}) = \{v \in \Sigma^* \mid \exists q \in Q : (q_0, v, Z_0) \vdash^* (q, \varepsilon, \varepsilon)\}$ *where* $\vdash^*$ *is the reflexive transitive closure of* $\vdash$*.*

3 μ-Regular Expressions

Regular expressions can be extended with a least fixed point operator μ to extend their scope to context-free languages [11].

Definition 3. *The set* $\mathbf{R}(\Sigma, X)$ *of* μ*-regular pre-expressions over alphabet* Σ *and set of variables* X *is defined as the smallest set such that*

- $\mathbf{0} \in \mathbf{R}(\Sigma, X)$,
- $\mathbf{1} \in \mathbf{R}(\Sigma, X)$,
- $a \in \Sigma$ *implies* $a \in \mathbf{R}(\Sigma, X)$,
- $r, s \in \mathbf{R}(\Sigma, X)$ *implies* $r \cdot s \in \mathbf{R}(\Sigma, X)$,
- $r, s \in \mathbf{R}(\Sigma, X)$ *implies* $r + s \in \mathbf{R}(\Sigma, X)$,
- $r \in \mathbf{R}(\Sigma, X)$ *implies* $r^* \in \mathbf{R}(\Sigma, X)$,
- $x \in X$ *implies* $x \in \mathbf{R}(\Sigma, X)$,
- $r \in \mathbf{R}(\Sigma, X \cup \{x\})$ *implies* $\mu x.r \in \mathbf{R}(\Sigma, X)$.

The set $\mathbf{R}(\Sigma)$ *of* μ*-regular expressions over* Σ *is defined as* $\mathbf{R}(\Sigma) := \mathbf{R}(\Sigma, \emptyset)$.

As customary, we consider the elements of $\mathbf{R}(\Sigma, X)$ as abstract syntax trees and freely use parentheses to disambiguate. We further assume that $*$ has higher precedence than $\cdot$, which has higher precedence than $+$. The μx-operator binds the recursion variable x with lowest precedence: its scope extends as far to the right as possible. A variable x occurs free if there is no enclosing μx-operator. A *closed* expression has no free variables.

Definition 4. *The language denoted by a* μ*-regular pre-expression is defined inductively by* $\mathcal{L} : \mathbf{R}(\Sigma, X) \times (X \to \wp(\Sigma^*)) \to \wp(\Sigma^*)$*. Let* $\eta \in X \to \wp(\Sigma^*)$ *be a mapping from variables to languages.*

- $\mathcal{L}(\mathbf{0}, \eta) = \{\}$.
- $\mathcal{L}(\mathbf{1}, \eta) = \{\varepsilon\}$.
- $\mathcal{L}(a, \eta) = \{a\}$ *(singleton letter word) for each* $a \in \Sigma$.
- $\mathcal{L}(r \cdot s, \eta) = \mathcal{L}(r, \eta) \cdot \mathcal{L}(s, \eta)$.
- $\mathcal{L}(r + s, \eta) = \mathcal{L}(r, \eta) \cup \mathcal{L}(s, \eta)$.
- $\mathcal{L}(r^*, \eta) = (\mathcal{L}(r, \eta))^*$.
- $\mathcal{L}(x, \eta) = \eta(x)$.
- $\mathcal{L}(\mu x.r, \eta) = \mathsf{lfp}\ L.\mathcal{L}(r, \eta[x \mapsto L])$.

For an expression $r \in \mathbf{R}(\Sigma)$*, we write* $\mathcal{L}(r) := \mathcal{L}(r, \emptyset)$.

Here, lfp is the *least fixed point operator* on the complete lattice $\wp(\Sigma^*)$ (ordered by set inclusion). Its application in the definition yields the smallest set $L \subseteq \Sigma^*$ such that $L = \mathcal{L}(r, \eta[x \mapsto L])$. This fixed point exists by Tarski's theorem because $\mathcal{L}$ is a monotone function, which is captured precisely in the following lemma.

Lemma 5. *For each finite set* X*,* $\eta \in X \to \wp(\Sigma)$*,* $r \in \mathbf{R}(\Sigma, X \cup \{x\})$*, the function* $L \mapsto \mathcal{L}(r, \eta[x \mapsto L])$ *is monotone on* $\wp(\Sigma^*)$*. That is, if* $L \subseteq L'$*, then* $\mathcal{L}(r, \eta[x \mapsto L]) \subseteq \mathcal{L}(r, \eta[x \mapsto L'])$.

According to Leiss [11], it is a folkore theorem that the languages generated by μ-regular expressions are exactly the context-free languages.

Theorem 6. $L \subseteq \Sigma^*$ *is context-free if and only if there exists a* μ*-regular expression* $r \in \mathbf{R}(\Sigma)$ *such that* $L = \mathcal{L}(r)$.

Subsequently we will deal syntactically with fixed points. To this end, we define properties of expressions and substitutions to make substitution application well-defined.

Definition 7. *Let* $\mathcal{X}$ *be the universe of variables occurring in expressions equipped with a strict partial order* $\prec$.

An expression is order-respecting *if each subexpression of the form* $\mu x.r$ *has only free variables which are strictly before* x*:* $\forall y \in fv(\mu x.r), y \prec x$.

A mapping $\sigma : X \to \mathbf{R}(\Sigma, X)$ *is* order-closed *if* $\forall x \in X$*,* $\sigma(x)$ *is order-respecting and* $\forall y \in fv(\sigma(x))$*,* $y \prec x$ *and* $y \in dom(\sigma)$.

A variable ordering for an expression always exists: assume that all binders bind different variables and take the topological sort of the subexpression containment.

We define the application $\sigma \bullet r$ of an order-closed mapping σ to an order-respecting expression r by starting to substitute a maximal free variable by its image and repeat this process until all variables are eliminated.

Definition 8. *Let* $X \subseteq \mathcal{X}$ *a finite set of variables,* $r \in \mathbf{R}(\Sigma, X)$ *order-respecting, and* $\sigma : X \to \mathbf{R}(\Sigma, X)$ *be order-closed.*

The application $\sigma \bullet r \in \mathbf{R}(\Sigma, X)$ *yields an expression that is defined by substituting for the free variables in* r *in descending order.*

$$\sigma \bullet r = \begin{cases} r & fv(r) = \emptyset \\ \sigma \bullet r[\sigma(x)/x] & x \in \max(fv(r)) \text{ is a maximal element} \end{cases}$$

Application is well-defined because the variables x are drawn from the finite set X and the substitution step for x only introduces new variables that are strictly smaller than x due to order-closedness. The outcome does not depend on the choice of the maximal variable because the unfolding of a maximal variable cannot contain one of the other maximal variables. Furthermore, all intermediate expressions (and thus the result) are order-respecting.

4 Partial Derivatives

Antimirov [4] introduced partial derivatives to study the syntactic transformation from regular expressions to nondeterministic and deterministic finite automata. A partial derivative $\partial_a(r)$ with respect to an input symbol a maps an expression r to a set of expressions such that their union denotes the left quotient of $\mathcal{L}(r)$. Antimirov's definition corresponds to the left part of Fig. 1. We write $\mathbf{R}_o(\Sigma)$ for the set of ordinary regular expressions that neither contain the μ-operator nor any variables. We extend $\cdot$ to a function $(\cdot) : \wp(\mathbf{R}(\Sigma, X)) \times \mathbf{R}(\Sigma, X) \to \wp(\mathbf{R}(\Sigma, X))$ on sets of expressions R defined pointwise by

$$R \cdot s = \{r \cdot s \mid r \in R\}.$$

The definition of partial derivatives relies on nullability, which is tested by a function $\mathcal{N} : \mathbf{R}_o(\Sigma) \to \mathbb{B}$. The right side of the figure corresponds to Antimirov's definition.

Lemma 9. *For all $r \in \mathbf{R}_o(\Sigma)$, $\mathcal{N}(r)$ iff $\varepsilon \in \mathcal{L}(r)$.*

Theorem 10 (Correctness [4]). *For all $r \in \mathbf{R}_o(\Sigma)$, $a \in \Sigma$, $\mathcal{L}(\partial_a(r)) = a\backslash\mathcal{L}(r)$.*

Here we adopt the convention that if R is a set of expressions, then $\mathcal{L}(R)$ denotes the union of the languages of all expressions: $\mathcal{L}(R) = \bigcup\{\mathcal{L}(r) \mid r \in R\}$.

Theorem 11 (Expansion). *For $r \in \mathbf{R}_o(\Sigma)$, $\mathcal{L}(r) = \{\varepsilon \mid \mathcal{N}(r)\} \cup \bigcup_{a\in\Sigma} a \cdot \mathcal{L}(\partial_a(r))$.*

$$\begin{aligned}
\partial_a(\mathbf{0}) &= \{\} & \mathcal{N}(\mathbf{0}) &= \mathbf{ff}\\
\partial_a(\mathbf{1}) &= \{\} & \mathcal{N}(\mathbf{1}) &= \mathbf{tt}\\
\partial_a(b) &= \{\mathbf{1} \mid a = b\} & \mathcal{N}(a) &= \mathbf{ff}\\
\partial_a(r + s) &= \partial_a(r) \cup \partial_a(s) & \mathcal{N}(r + s) &= \mathcal{N}(r) \vee \mathcal{N}(s)\\
\partial_a(r \cdot s) &= \partial_a(r) \cdot s \cup \{s' \mid \mathcal{N}(r), s' \in \partial_a(s)\} & \mathcal{N}(r \cdot s) &= \mathcal{N}(r) \wedge \mathcal{N}(s)\\
\partial_a(r^*) &= \partial_a(r) \cdot r^* & \mathcal{N}(r^*) &= \mathbf{tt}
\end{aligned}$$

Fig. 1. Antimirov's definition of partial derivatives and nullability

Partial derivatives give rise to a nondeterministic finite automaton.

Theorem 12 (Finiteness [4]). *Let $r \in \mathbf{R}_o(\Sigma)$ be a regular expression. Define partial derivatives by words by $\partial_\varepsilon(r) = \{r\}$ and $\partial_{aw}(r) = \bigcup\{\partial_w(s) \mid s \in \partial_a(r)\}$ and by a language L by $\partial_L(r) = \bigcup\{\partial_w(r) \mid w \in L\}$.*

The set $\partial_{\Sigma^*}(r)$ is finite.

Theorem 13 (Nondeterministic finite automaton construction [4]). *Let $r \in \mathbf{R}_o(\Sigma)$ be a regular expression and define $Q = \partial_{\Sigma^*}(r)$, $\delta : Q \times \Sigma \to \wp(Q)$ by $(q, a, q') \in \delta$ iff $q' \in \partial_a(q)$. Let further $q_0 = r$ and $F = \{q \in Q \mid \mathcal{N}(q)\}$.*

Then $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ is a NFA such that $\mathcal{L}(r) = \mathcal{L}(\mathcal{A})$.

The plan is to extend these results to μ-regular expressions. We start with the extension of the nullability function.

5 Nullability

Figure 2 extends nullability to μ-regular expressions. To cater for recursion, the $\mathcal{N}$ function obtains as a further argument a nullability environment ν of type $X \to \mathbb{B}$. With this extension, an expression $\mu x.r$ is deemed nullable if its body r is nullable. Furthermore, the least fixed point operator feeds back the nullability of the body to the free occurrences of the recursion variables. This fixed point is computed on the two-element Boolean lattice $\mathbb{B}$ ordered by $\mathbf{ff} \sqsubseteq \mathbf{tt}$ with disjunction $(\vee) : \mathbb{B} \times \mathbb{B} \to \mathbb{B}$ as the least upper bound operation. Thus, the case for a free variable x obtains its nullability information from the nullability environment.

Lemma 14. *For each $r \in \mathbf{R}(\Sigma, X)$, $\mathcal{N}(r)$ is a monotone function from $X \to \mathbb{B}$ (ordered pointwise) to $\mathbb{B}$.*

To prepare for the correctness proof of $\mathcal{N}$, we first simplify the case for the fixed point. It turns out that one iteration is sufficient to obtain the fixed point. This fact is also a consequence of a standard result, namely that the number

$$
\begin{aligned}
\mathcal{N}(\mathbf{0})\nu &= \mathbf{ff} \\
\mathcal{N}(\mathbf{1})\nu &= \mathbf{tt} \\
\mathcal{N}(a)\nu &= \mathbf{ff} \\
\mathcal{N}(r + s)\nu &= \mathcal{N}(r)\nu \vee \mathcal{N}(s)\nu \\
\mathcal{N}(r \cdot s)\nu &= \mathcal{N}(r)\nu \wedge \mathcal{N}(s)\nu \\
\mathcal{N}(r^*)\nu &= \mathbf{tt} \\
\mathcal{N}(\mu x.r)\nu &= \mathsf{lfp}\ b.\mathcal{N}(r)\nu[x \mapsto b] \\
\mathcal{N}(x)\nu &= \nu(x)
\end{aligned}
$$

Fig. 2. Nullability of μ-regular expressions

of iterations needed to compute the fixed point of a monotone function on a lattice is bounded by the height of the lattice. In this case, the Boolean lattice has height one.

Lemma 15. *Let X be a set of variables, $r \in \mathbf{R}(\Sigma, X \cup \{x\})$, $\eta : X \to \wp(\Sigma^*)$, and $L \subseteq \Sigma^*$ such that $\varepsilon \notin L$. If $\varepsilon \notin \mathcal{L}(r, \eta[x \mapsto \emptyset])$, then $\varepsilon \notin \mathcal{L}(r, \eta[x \mapsto L])$.*

Lemma 16. *For all $r \in \mathbf{R}(\Sigma, X)$, for all $\nu : X \to \mathbb{B}$,*

$$\mathit{lfp}\ b.\mathcal{N}(r)\nu[x \mapsto b] = \mathcal{N}(r)\nu[x \mapsto \mathbf{ff}].$$

For the statement of the correctness, we need to define what it means for a nullability environment to agree with a language environment.

Definition 17. *Nullability environment $\nu : X \to \mathbb{B}$* agrees *with language environment $\eta : X \to \wp(\Sigma^*)$, written $\eta \models \nu$, if for all $x \in X$, $\varepsilon \in \eta(x)$ iff $\nu(x)$.*

Lemma 18 (Correctness of $\mathcal{N}$). *For all X, $r \in \mathbf{R}(\Sigma, X)$, $\eta \in X \to \wp(\Sigma^*)$, $\nu \in X \to \mathbb{B}$, such that $\eta \models \nu$, it holds that $\varepsilon \in \mathcal{L}(r, \eta)$ iff $\mathcal{N}(r)\nu$.*

6 Derivation

The derivative for μ-regular expressions has a **different type** than for ordinary regular expressions: A partial derivative is a set of **non-empty sequences** (i.e., stack fragments) of regular expressions. The idea is that deriving a recursion operator $\mu x.r$ pushes the current context on the stack and starts afresh with the derivation of r. In other words, the derivative function for μ-regular expressions has the same signature as the transition function for a nondeterministic PDA.

To distinguish operations on stacks from operations on words over Σ, we write ":" (read "push") for the concatenation operator on stacks. We also use this operator for pattern matching parts of a stack. We write $[\,]$ for the empty stack, $[r_1, \ldots, r_n]$ for a stack with n elements, and $\overline{\mathbf{r}}$ for any stack of expressions. We extend the concatenation operator for regular expressions to non-empty stacks by having it operate on the **last** (bottom) element of a stack.

$$\begin{aligned}
\partial_\alpha^{\sigma,\nu}(\mathbf{0}) &= \{\} \\
\partial_\alpha^{\sigma,\nu}(\mathbf{1}) &= \{\} \\
\partial_\alpha^{\sigma,\nu}(b) &= \{[\mathbf{1}] \mid \alpha = b \in \Sigma\} \\
\partial_\alpha^{\sigma,\nu}(r + s) &= \partial_\alpha^{\sigma,\nu}(r) \cup \partial_\alpha^{\sigma,\nu}(s) \\
\partial_\alpha^{\sigma,\nu}(r \cdot s) &= \partial_\alpha^{\sigma,\nu}(r) \cdot (\sigma \bullet s) \cup \{\overline{\mathbf{s}} \mid \mathcal{N}(r)\nu, \overline{\mathbf{s}} \in \partial_\alpha^{\sigma,\nu}(s)\} \\
\partial_\alpha^{\sigma,\nu}(r^*) &= \partial_\alpha^{\sigma,\nu}(r) \cdot (\sigma \bullet r^*) \\
\partial_\alpha^{\sigma,\nu}(\mu x.r) &= \partial_\alpha^{\sigma[\mu x.r/x],\nu[\mathcal{N}(r)\nu[\mathbf{ff}/x]/x]}(r) : [\mathbf{1}] \\
\partial_\alpha^{\sigma,\nu}(x) &= \{[\sigma \bullet x] \mid \alpha = \varepsilon\}
\end{aligned}$$

Fig. 3. Partial derivatives of μ-regular expressions for $\alpha \in \Sigma \cup \{\varepsilon\}$

Definition 19. *Let $(M, (\cdot), \mathbf{1})$ be a monoid. We lift the monoid operation to non-empty stacks $(\cdot) \in M^+ \times M \to M^+$ for $\overline{a} \in M^*$ and $a, b \in M$ by*

$$(\overline{a} : [a]) \cdot b = (\overline{a} : [a \cdot b]).$$

We further lift it pointwise to sets $A \subseteq M^+$ to obtain $(\cdot) \in \wp(M^+) \times M \to \wp(M^+)$:

$$A \cdot b = \{\overline{a} \cdot b \mid \overline{a} \in A\}.$$

We use this definition for $M = \mathbf{R}(\Sigma, X)$ and also extend the push operation $(:)$ pointwise to sets of stacks.

$$\begin{aligned}(:) &\in \wp(\mathbf{R}(\Sigma, X)^+) \times \mathbf{R}(\Sigma, X)^+ \to \wp(\mathbf{R}(\Sigma, X)^+) \\ R : \overline{\mathbf{s}} &= \{\overline{\mathbf{r}} : \overline{\mathbf{s}} \mid \overline{\mathbf{r}} \in R\}\end{aligned}$$

Most of the time, the second argument will be a singleton stack $[s]$.

Before we discuss the intricacies of the full definition in Fig. 3, let's first consider a naive extension of the derivative function in Fig. 1 to μ-regular expressions and analyze its problems:

$$\partial_a(\mu x.r) = \partial_a(r[\mu x.r/x]) : [\mathbf{1}] \qquad \text{(naive unrolling: to be revised)}$$

Taking the derivative of a recursive definition means to apply the derivative to the unrolled definition. At the same time, we push an empty context on the stack so that the context of the recursion does not become a direct part of the derivative. This proposed definition makes sure that the partial derivative $\partial_a(r)$ is only ever applied to closed expressions $r \in \mathbf{R}(\Sigma)$. Hence, the case of a free recursion variable x would not occur during the computation of $\partial_a(r)$.

Example 20. The "naive unrolling" definition of the partial derivative has a problem. While it can be shown to be (partially) correct, it is not well-defined for all arguments. Consider the left-recursive expression $r = \mu x.\mathbf{1} + x \cdot a$, which is equivalent to a^*. Computing its partial derivative according to "naive unrolling" reveals that it depends on itself, so that $\partial_a(r)$ would be undefined.

$$\begin{aligned}\underline{\partial_a(r)} &= \partial_a(\mathbf{1} + r \cdot a) : [\mathbf{1}] \\ &= (\{\} \cup \partial_a(r \cdot a)) : [\mathbf{1}] \\ &= (\partial_a(r) \cdot a \cup \partial_a(a)) : [\mathbf{1}] \\ &= (\underline{\partial_a(r)} \cdot a \cup \{[\mathbf{1}]\}) : [\mathbf{1}]\end{aligned}$$

We remark that the expression r corresponds to a left-recursive grammar, where the naive construction of a top-down parser using the method of recursive descent also runs into problems [2]. There would be no problem with the right-recursive equivalent $r' = \mu x.\mathbf{1} + a \cdot x$ where the naive unrolling yields $\partial_a(r') = \{[r', \mathbf{1}]\}$. Indeed, the work by Winter and others [19] only allows guarded uses of the recursion operator, which rules out expressions like r from the start and which enables them to use the "naive unrolling" definition of the derivative.

For that reason, the derivative must not simply unroll recursions as they are encountered. Our definition distinguishes between left-recursive occurrences of a recursion variable, which must not be unrolled, and guarded occurrences, which can be unrolled safely. The derivative function remembers deferred unrollings in a substitution σ and applies them only when it is safe.

These observations lead to the signature of the definition of partial derivative in Fig. 3. Its type is

$$\partial : (\Sigma \cup \{\varepsilon\}) \times (X \to \mathbf{R}(\Sigma, X)) \times (X \to \mathbb{B}) \times \mathbf{R}(\Sigma, X) \to \wp(\mathbf{R}(\Sigma)^+)$$

and we write it as $\partial_\alpha^{\sigma,\nu}(r)$. It takes a symbol or an empty string $\alpha \in \Sigma \cup \{\varepsilon\}$ to derive, a substitution $\sigma : X \to \mathbf{R}(\Sigma, X)$ that maps free recursion variables to expressions (i.e., their unrollings), a nullability function $\nu : X \to \mathbb{B}$ that maps free recursion variables to their nullability, and the regular expression $r \in \mathbf{R}(\Sigma, X)$ to derive as arguments and returns the partial derivatives as a set of non-empty stacks of expressions.

Let's examine how the revised definition guarantees well-definedness. Example 20 demonstrates that left recursion is the cause for non-termination of the naive definition. The problem is that the naive definition indiscriminately substitutes all occurrences of x by its unfolding and propagates the derivative into the unfolding. However, this substitution is only safe in guarded positions (i.e., behind at least one terminal symbol in the unfolding). To avoid substitution in unguarded positions, the definition in Fig. 3 reifies this substitution as an additional argument σ and takes care to only apply it in guarded positions.

To introduce this recursion, the derived symbol α ranges over $\Sigma \cup \{\varepsilon\}$ in Fig. 3. For $\alpha = \varepsilon$, the derivative function unfolds one step of left recursion.

Example 21. Recall $r = \mu x.\mathbf{1} + x \cdot a$ from Example 20. Observe that $\mathcal{N}(r)\emptyset = \mathcal{N}(\mathbf{1} + x \cdot a)[\mathbf{ff}/x] = \mathbf{tt}$.

$$\begin{aligned}
\partial_a^{\emptyset,\emptyset}(r) &= (\partial_a^{[r/x],[\mathbf{tt}/x]}(\mathbf{1} + x \cdot a)) : [\mathbf{1}] \\
&= (\{\} \cup \partial_a^{[r/x],[\mathbf{tt}/x]}(x \cdot a)) : [\mathbf{1}] \\
&= (\{[\mathbf{1}]\}) : [\mathbf{1}] \\
&= \{[\mathbf{1}, \mathbf{1}]\}
\end{aligned}$$

The spontaneous derivative unfolds one level of left recursion.

$$\begin{aligned}
\partial_\varepsilon^{\emptyset,\emptyset}(r) &= (\partial_\varepsilon^{[r/x],[\mathbf{tt}/x]}(\mathbf{1} + x \cdot a)) : [\mathbf{1}] \\
&= (\{\} \cup \partial_\varepsilon^{[r/x],[\mathbf{tt}/x]}(x \cdot a)) : [\mathbf{1}] \\
&= (\{[r \cdot a]\}) : [\mathbf{1}] \\
&= \{[r \cdot a, \mathbf{1}]\}
\end{aligned}$$

Thus, the spontaneous derivative corresponds to ε-transitions of the PDA that is to be constructed.

7 Correctness

To argue about the correctness of our derivative operation, we define the membership of a word $w \in \Sigma^*$ in the language of an order-respecting expression $r \in \mathbf{R}(\Sigma, X)$ under an order-closed mapping $\sigma : X \to \mathbf{R}(\Sigma, X)$ inductively by the judgment $\sigma \vdash w \in r$ in Fig. 4 along with $\sigma \vdash w \in \overline{\mathbf{r}}$ for an expression stack $\overline{\mathbf{r}}$ and $\sigma \vdash w \in R$ for a set of such stacks $R \subseteq \mathbf{R}(\Sigma, X)^*$. This inductive definition mirrors the previous fixed point definition of the language of an expression.

Lemma 22. *For all $r \in \mathbf{R}(\Sigma)$ and $w \in \Sigma^*$. $\emptyset \vdash w \in r$ iff $w \in \mathcal{L}(r)$.*

It is straightforward to prove the following derived rule.

$$\boxed{\sigma \vdash w \in r} \qquad \sigma \vdash \varepsilon \in \mathbf{1} \qquad \sigma \vdash a \in a \qquad \frac{\sigma \vdash w \in r}{\sigma \vdash w \in r + s} \qquad \frac{\sigma \vdash w \in s}{\sigma \vdash w \in r + s}$$

$$\frac{\sigma \vdash v \in r \qquad \sigma \vdash w \in s}{\sigma \vdash vw \in r \cdot s} \qquad \sigma \vdash \varepsilon \in r^* \qquad \frac{\sigma \vdash v \in r \qquad \sigma \vdash w \in r^*}{\sigma \vdash vw \in r^*}$$

$$\frac{\sigma[\mu x.r/x] \vdash w \in r}{\sigma \vdash w \in \mu x.r} \qquad \frac{\sigma \vdash w \in \mu x.r}{\sigma[\mu x.r/x] \vdash w \in x}$$

$$\boxed{\sigma \vdash w \in \overline{\mathbf{r}}} \qquad \sigma \vdash \varepsilon \in [] \qquad \frac{\sigma \vdash v \in r \qquad \sigma \vdash w \in \overline{\mathbf{r}}}{\sigma \vdash vw \in [r] : \overline{\mathbf{r}}}$$

$$\boxed{\sigma \vdash w \in R} \qquad \frac{\sigma \vdash w \in \overline{\mathbf{r}} \qquad \overline{\mathbf{r}} \in R}{\sigma \vdash w \in R}$$

Fig. 4. Membership in a μ-regular expression, a stack of expressions, and a set of stacks

Lemma 23. *If $R \subseteq S \subseteq \mathbf{R}(\Sigma, X)^*$, then $\sigma \vdash w \in R$ implies $\sigma \vdash w \in S$.*

Lemma 24. *Let $r \in \mathbf{R}(\Sigma, X)$ and $\sigma : X \to \mathbf{R}(\Sigma, X)$ be order-respecting. If $\sigma \vdash w \in r$, then $\emptyset \vdash w \in \sigma \bullet r$.*

The derivation closure $\tilde{\partial}_a(\overline{\mathbf{r}})$ of a non-empty closed stack of expressions is defined by the union of the partial derivatives after taking an arbitrary number of ε-steps. It is our main tool in proving the correctness of the derivative.

Definition 25. *For $a \in \Sigma$, the derivation closure $\tilde{\partial}_a^{\sigma,\nu}(r : \overline{\mathbf{r}})$ is inductively defined as the smallest set of stacks such that*

1. $\tilde{\partial}_a^{\sigma,\nu}(r : \overline{\mathbf{r}}) \supseteq \partial_a^{\sigma,\nu}(r) : \overline{\mathbf{r}}$ *and*
2. $\tilde{\partial}_a^{\sigma,\nu}(r : \overline{\mathbf{r}}) \supseteq \bigcup\{\tilde{\partial}_a^{\sigma,\nu}(\overline{\mathbf{s}} : \overline{\mathbf{r}}) \mid \overline{\mathbf{s}} \in \partial_\varepsilon^{\sigma,\nu}(r)\}$.

Lemma 26 (Unfolding). *Let $r \in \mathbf{R}(\Sigma, X)$ an order-respecting expression, $\sigma : X \to \mathbf{R}(\Sigma, X)$ order-closed with $\sigma(x) = \mu x.s_x$, $\nu : X \to \mathbb{B}$ such that $\nu(x) = \mathcal{N}(\sigma \bullet x)\emptyset$.*

$$\sigma \vdash w \in r \Leftarrow \emptyset \vdash w \in \partial_\varepsilon^{\sigma,\nu}(r)$$

Theorem 27 (Correctness). *Let $r \in \mathbf{R}(\Sigma, X)$ an order-respecting expression, $\sigma : X \to \mathbf{R}(\Sigma, X)$ order-closed with $\sigma(x) = \mu x.s_x$, $\nu : X \to \mathbb{B}$ such that $\nu(x) = \mathcal{N}(\sigma \bullet x)\emptyset$.*

$$\sigma \vdash aw \in r \Leftrightarrow \emptyset \vdash w \in \tilde{\partial}_a^{\sigma,\nu}([r])$$

Proof. The direction from left to right is proved by induction on $\sigma \vdash aw \in r$.

We demonstrate the right-to-left direction.

Suppose that $\Delta = \emptyset \vdash w \in \tilde{\partial}_a^{\sigma,\nu}([r])$ and show that $\sigma \vdash aw \in r$.

The proof is by induction on the size of the derivation of Δ. Inversion yields that there is some $\overline{\mathbf{r}} \in \tilde{\partial}_a^{\sigma,\nu}([r])$ such that $\emptyset \vdash w \in \overline{\mathbf{r}}$. Now there are two cases.

Case $w = \varepsilon$ and $\overline{\mathbf{r}} = []$ so that the empty-sequence-rule $\emptyset \vdash \varepsilon \in []$ applies. But this case cannot happen because $\overline{\mathbf{r}} \neq []$.

Case $\emptyset \vdash vw \in [s] : \overline{\mathbf{r}}$ because $\emptyset \vdash v \in s$ and $\emptyset \vdash w \in \overline{\mathbf{r}}$.

These two cases boil down to $w = w_1 \ldots w_n$, $\overline{\mathbf{r}} = [r_1, \ldots, r_n]$, for some $n \geq 1$, and $\emptyset \vdash w_1 \ldots w_n \in [r_1, \ldots, r_n]$ because $\emptyset \vdash w_i \in r_i$.

We perform an inner induction on r.

Case $\mathbf{0}$, $\mathbf{1}$, $b \neq a$: contradictory because $\tilde{\partial}_a^{\sigma,\nu}([r]) = \emptyset$.

Case a: $\tilde{\partial}_a^{\sigma,\nu}([a]) = \{[\mathbf{1}]\}$ so that $w = \varepsilon$. Clearly, $\sigma \vdash a \in a$.

Case $r + s$: We can show that $\tilde{\partial}_a^{\sigma,\nu}(r + s) = \tilde{\partial}_a^{\sigma,\nu}(r) \cup \tilde{\partial}_a^{\sigma,\nu}(s)$. Assuming that $\overline{\mathbf{r}} \in \tilde{\partial}_a^{\sigma,\nu}(r)$, induction on r yields $\sigma \vdash aw \in r$ and the $+$-rule yields $\sigma \vdash aw \in r+s$. Analogously for s.

Case $r \cdot s$: We can show that $\tilde{\partial}_a^{\sigma,\nu}(r \cdot s) = \tilde{\partial}_a^{\sigma,\nu}(r) \cdot (\sigma \bullet s) \cup \{\overline{\mathbf{s}} \mid \mathcal{N}(r)\nu, \overline{\mathbf{s}} \in \tilde{\partial}_a^{\sigma,\nu}(s)\}$. There are two cases.

Subcase $\overline{\mathbf{r}} \in \tilde{\partial}_a^{\sigma,\nu}(r) \cdot (\sigma \bullet s)$. Hence, $\overline{\mathbf{r}} = [r_1, \ldots, r_n \cdot (\sigma \bullet s)]$ so that $w = w_1 \ldots w_n w_{n+1}$ and $\emptyset \vdash w_1 \in r_1$, $\ldots$, $\emptyset \vdash w_n \in r_n$, and $\emptyset \vdash w_{n+1} \in (\sigma \bullet s)$. Now, $\overline{\mathbf{r}}' = [r_1, \ldots, r_n] \in \tilde{\partial}_a^{\sigma,\nu}(r)$ and thus $\emptyset \vdash w_1 \ldots w_n \in \tilde{\partial}_a^{\sigma,\nu}(r)$. By induction, $\sigma \vdash aw_1 \ldots w_n \in r$. Because $\sigma \bullet s$ is closed, we also have $\emptyset \vdash w_{n+1} \in (\sigma \bullet s)$ and thus by Lemma 24 $\sigma \vdash w_{n+1} \in s$. Taken together $\sigma \vdash aw_1 \ldots w_n w_{n+1} \in r \cdot s$.

Subcase $\mathcal{N}(r)\nu$ and $\overline{\mathbf{r}} \in \partial_a^{\sigma,\nu}(s)$. Hence, $\sigma \vdash \varepsilon \in r$, by induction $\sigma \vdash aw \in s$, and the concatenation rule yields $\sigma \vdash aw \in r \cdot s$.

Case r^*. Because $\overline{\mathbf{r}} \in \tilde{\partial}_a^{\sigma,\nu}(r) \cdot (\sigma \bullet r^*)$, it must be that $\overline{\mathbf{r}} = [r_1, \ldots, r_n \cdot (\sigma \bullet r^*)]$ and $w = w_1 \ldots w_n w_{n+1}$ so that $\emptyset \vdash w_1 \in r_1, \ldots, \emptyset \vdash w_n \in r_n$, and $\emptyset \vdash w_{n+1} \in (\sigma \bullet r^*)$. Proceed as in the first subcase for concatenation.

Case $\mu x.r$. As usual, let $\hat{\sigma} = \sigma[\mu x.r/x]$ and $\hat{\nu} = \nu[\mathcal{N}(r)\nu[\mathbf{ff}/x]/x]$. Again, $\tilde{\partial}_a^{\sigma,\nu}(\mu x.r) = \tilde{\partial}_a^{\hat{\sigma},\hat{\mu}}(r) : [\mathbf{1}]$. Hence, $\overline{\mathbf{r}} = \overline{\mathbf{r}}' : [\mathbf{1}]$ for some $\overline{\mathbf{r}}' \in \tilde{\partial}_a^{\hat{\sigma},\hat{\mu}}(r)$ such that $\emptyset \vdash w \in \overline{\mathbf{r}}'$. Induction yields that $\hat{\sigma} \vdash aw \in r$ and application of the μ-rule yields $\sigma \vdash aw \in \mu x.r$.

Case x. Then $\tilde{\partial}_a^{\hat{\sigma},\hat{\nu}}(x) = \tilde{\partial}_a^{\sigma,\nu}(\mu x.r)$ if $\hat{\sigma} = \sigma[\mu x.r/x]$ and $\hat{\nu} = \nu[\mathcal{N}(r)\nu[\mathbf{ff}/x]/x]$.

Now $\emptyset \vdash w \in \tilde{\partial}_a^{\hat{\sigma},\hat{\nu}}(x)$ iff exists $\overline{\mathbf{r}} \in \tilde{\partial}_a^{\hat{\sigma},\hat{\nu}}(x) = \tilde{\partial}_a^{\sigma,\nu}(\mu x.r)$ such that $\emptyset \vdash w \in \overline{\mathbf{r}}$. But $\tilde{\partial}_a^{\sigma,\nu}(\mu x.r) = \tilde{\partial}_a^{\hat{\sigma},\hat{\nu}}(r) : [\mathbf{1}]$ so that $\overline{\mathbf{r}} = \overline{\mathbf{r}}' : [\mathbf{1}]$ and $\emptyset \vdash w \in \overline{\mathbf{r}}'$ with a smaller derivation tree. Thus, induction yields that $\hat{\sigma} \vdash aw \in r$, application of the μ-rule yields $\sigma \vdash aw \in \mu x.r$, and application of the variable rule yields $\hat{\sigma} \vdash aw \in x$, as desired. □

8 Finiteness

In analogy to Antimirov's finite automaton construction, we aim to use the set of iterated derivatives as a building block for a pushdown automaton. In our construction, derivatives end up as pushdown symbols rather than states: the top of the pushdown plays the role of the state. It remains to prove that this set is finite to obtain a proper PDA.

Our finiteness argument is based on an analysis of the syntactical form of the derivatives. It turns out that a derivative is, roughly, a concatenation of a strictly descending sequence of certain subexpressions of the initial expression. As this ordering is finite, we obtain a finite bound on the syntactically possible derivatives.

We start with an analysis of the output of $\partial_\alpha^{\sigma,\nu}(r)$. The elements in the stack of a partial derivative are vectors of the form $((h \cdot s_1) \cdot s_2) \cdots s_k$ that we abbreviate $h \cdot \vec{s}$, where the s_i are arbitrary expressions and h is either $\mathbf{1}$ or $\mu x.r$ where $\mu x.r$ is closed.

It turns out that the vectors produced by derivation are always strictly ascending chains in the subterm ordering of the original expression, say t. We first define this ordering, then we define the structure of these vectors in Definition 31.

Definition 28. *Let $r \in \mathbf{R}(\Sigma)$ be a closed expression. We define the* addressing function $A_r : \mathbb{N}^* \hookrightarrow \mathbf{R}(\Sigma, X)$ *by induction on r.*

$$
\begin{aligned}
A_{\mathbf{0}} &= \{(\varepsilon, \mathbf{0})\} & A_{r+s} &= \{(\varepsilon, r+s)\} \cup 1.A_r \cup 2.A_s \\
A_{\mathbf{1}} &= \{(\varepsilon, \mathbf{1})\} & A_{r \cdot s} &= \{(\varepsilon, r \cdot s)\} \cup 1.A_r \cup 2.A_s \\
A_a &= \{(\varepsilon, a)\} & A_{r^*} &= \{(\varepsilon, r^*)\} \cup 1.A_r \\
A_x &= \{(\varepsilon, x)\} & A_{\mu x.r} &= \{(\varepsilon, \mu x.r)\} \cup 1.A_r
\end{aligned}
$$

Here $i.A$ modifies the function A by prepending i to each element of A's domain:

$$
(i.A)(w) = \begin{cases} A(w') & w = iw' \text{ and } A(w') \text{ defined} \\ \text{undefined} & \text{otherwise.} \end{cases}
$$

It is well known that $dom(A_r)$ is prefix-closed and assigns a unique $w \in \mathbb{N}^*$ to each occurrence of a subexpression in r. Let $r_1 = A_r(w_1)$ and $r_2 = A_r(w_2)$ be subexpression occurrences of r. We say that r_1 *occurs before* r_2 *in* r if $w_1 \preceq w_2$ in the lexicographic order on $\mathbb{N}^*$:

$$\varepsilon \preceq w \qquad \frac{i < j}{iv \preceq jw} \qquad \frac{v \preceq w}{iv \preceq iw}$$

We write $w_1 \prec w_2$ if $w_1 \preceq w_2$ and $w_1 \neq w_2$, in which case we say that r_1 occurs strictly before r_2.

Lemma 29. *For each closed expression $r \in \mathbf{R}(\Sigma)$, the strict lexicographic ordering $\prec$ on $dom(A_r)$ has no infinite chains.*

Definition 30. *Let $t \in \mathbf{R}(\Sigma)$ be a closed expression such that each variable occurring in t is bound exactly once. The* unfolding substitution σ_t *is defined by induction on t.*

$$\begin{aligned} \sigma_{\mathbf{0}} &= [] & \sigma_{r+s} &= \sigma_r \cup \sigma_s \\ \sigma_{\mathbf{1}} &= [] & \sigma_{r \cdot s} &= \sigma_r \cup \sigma_s \\ \sigma_a &= [] & \sigma_{r^*} &= \sigma_r \\ \sigma_x &= [] & \sigma_{\mu x.r} &= [\mu x.r/x] \cup \sigma_r \end{aligned}$$

Definition 31. *A vector $\vec{s} = (s_1 \cdot s_2) \cdots s_k$ is* t-sorted *if for all $1 \leq i < j \leq k$: s_i and s_j are subexpressions of t and s_i occurs strictly before s_j, which means that there are $w_1, \ldots, w_k \in \mathbb{N}^*$ such that $s_i = A_t(w_i)$ and $w_i \prec w_{i+1}$, for $1 \leq i < k$.*

For a t-sorted vector $\vec{s} = (s_1 \cdot s_2) \cdots s_k$ define two forms of expressions:

top: $\sigma_t \bullet (\mathbf{1} \cdot \vec{s})$.
rec: $\sigma_t \bullet ((\mu x.s) \cdot \vec{s})$ *where $\mu x.s$ is a subexpression of t and either $\mu x.s$ or an occurrence of x is strictly before s_i, for all $1 \leq i \leq k$.*

A stack $\overline{\mathbf{r}} = [r_1, \ldots, r_n]$ (for $n \geq 1$) has form $\mathbf{top}^+$ *if $r_1, \ldots, r_n$ have form* **top**.

A stack $\overline{\mathbf{r}} = [r_1, \ldots, r_n]$ (for $n \geq 1$) has form $\mathbf{rec}.\mathbf{top}^*$ *if r_1 has form* **rec** *and $r_2, \ldots, r_n$ have form* **top**.

Next, we show that all derivatives and partial derivatives of subexpressions of a closed expression t have indeed one of the forms $\mathbf{top}^+$ or $\mathbf{rec}.\mathbf{top}^*$.

Lemma 32 (Classification of derivatives). *Suppose that $t \in \mathbf{R}(\Sigma)$ is a closed expression, $r \in \mathbf{R}(\Sigma, X)$ is a subexpression of t, $\sigma : X \to \mathbf{R}(\Sigma, X)$ is order-closed with $\sigma(x) = \mu x.s$ (for $x \in X$ and $\mu x.s$ a subterm of t), and $\nu : X \to \mathbb{B}$ such that $\nu(x) = \mathcal{N}(\sigma \bullet x)\emptyset$. If $\overline{\mathbf{r}} = [r_1, \ldots, r_n] \in \partial_a^{\sigma,\nu}(r)$, then $n \geq 1$ and $\overline{\mathbf{r}}$ has form $\mathbf{top}^+$ and each $r_i = h_i \cdot \vec{s_i}$ for some t-sorted $\vec{s_i}$ which is before r.*

Lemma 33 (Classification of spontaneous derivatives). *Suppose that* $t \in \mathbf{R}(\Sigma)$ *is a closed expression,* $r \in \mathbf{R}(\Sigma, X)$ *is a subexpression of* t*,* $\sigma : X \to \mathbf{R}(\Sigma, X)$ *is order-closed with* $\sigma(x) = \mu x.s$ *(for* $x \in X$ *and* $\mu x.s$ *a subterm of* t*), and* $\nu : X \to \mathbb{B}$ *such that* $\nu(x) = \mathcal{N}(\sigma \bullet x)\emptyset$*. If* $\bar{\mathbf{r}} = [r_1, \ldots, r_n] \in \partial_{\varepsilon}^{\sigma,\nu}(r)$*, then* $n \geq 1$ *and* $\bar{\mathbf{r}}$ *has form* $\mathbf{rec}.\mathbf{top}^*$ *and each* $r_i = h_i \cdot \vec{s_i}$ *for some* t*-sorted* $\vec{s_i}$ *which is before* r*.*

Lemma 34 (Classification of derivatives of vectors). *Let* $t \in R(\Sigma)$ *be a closed expression and* t_0 *be closed of form* $\mathbf{top}$ *or form* $\mathbf{rec}$ *with respect to* t*. Then the elements of* $\partial_a^{\emptyset,\emptyset}(t_0)$ *are stacks of the form* $\mathbf{top}^+$ *as in Lemma 32 and the elements of* $\partial_{\varepsilon}^{\emptyset,\emptyset}(t_0)$ *are stacks of the form* $\mathbf{rec}.\mathbf{top}^*$*.*

We define the set of iterated partial derivatives as the expressions that may show up in the stack of a partial derivative. This set will serve as the basis for defining the set of pushdown symbols of a PDA.

Definition 35 (Iterated Partial Derivatives). *Let* $t \in \mathbf{R}(\Sigma)$ *be a closed expression. Define* $\Delta(t)$*, the set of iterated partial derivatives of* t*, as the smallest set such that*

- $\mathbf{1} \cdot t \in \Delta(t)$*;*
- *if* $r \in \Delta(t)$ *and* $[t_1, \ldots, t_n] \in \partial_a^{\emptyset,\emptyset}(r)$*, then* $t_j \in \Delta(t)$*, for all* $1 \leq j \leq n$*; and*
- *if* $r \in \Delta(t)$ *and* $[t_1, \ldots, t_n] \in \partial_{\varepsilon}^{\emptyset,\emptyset}(r)$*, then* $t_j \in \Delta(t)$*, for all* $1 \leq j \leq n$*.*

Lemma 36 (Closure). *Let* $t \in \mathbf{R}(\Sigma)$ *be a closed expression. Then all elements of* $\Delta(t)$ *either have form* $\mathbf{top}$ *or* $\mathbf{rec}$ *with respect to* t*.*

Proof. Follows from Lemmas 32, 33, and 34.

Lemma 37 (Finiteness). *Let* $t \in \mathbf{R}(\Sigma)$ *be closed. Then* $\Delta(t)$ *is finite.*

Proof. By construction, the elements of $\Delta(t)$ are all closed and have either form $\mathbf{top}$ or form $\mathbf{rec}$, which is a vector of the form $\sigma_t \bullet (h \cdot \vec{s})$ where $\vec{s}$ is t-sorted. As t is a finite expression and a t-sorted vector is strictly decreasing, there are only finitely many candidates for $\vec{s}$ (by Lemma 29).

The head h of the vector is either $\mathbf{1}$ or it is a subexpression of t of the form $\mu x.s_x$. Hence, there are only finitely many choices for h.

Thus $\Delta(t)$ is a subset of a finite set and hence finite. □

9 Automaton Construction

Given that the derivative for a closed μ-regular expression gives rise to a finite set of iterated partial derivatives, we use that set as the pushdown alphabet to construct a nondeterministic pushdown automaton that recognizes the same language. This construction is straightforward as its transition function corresponds exactly to the derivative and the spontaneous derivative function.

Definition 38. *Suppose that $t \in \mathbf{R}(\Sigma)$ is closed. Define the PDA $\mathcal{UA}(t) = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$ by a singleton set $Q = \{q\}$, $\Gamma = \Delta(t)$, $q_0 = q$, $Z_0 = \mathbf{1} \cdot r$, and $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \times Q \times \Gamma^*$ as the smallest relation such that*

- $(q, a, s, q, \overline{\mathbf{s}}) \in \delta$ *if* $\overline{\mathbf{s}} \in \partial_a^{\emptyset,\emptyset}(s)$, *for all* $s \in \Gamma$, $\overline{\mathbf{s}} \in \Gamma^*$, $a \in \Sigma$;
- $(q, \varepsilon, s, q, \overline{\mathbf{s}}) \in \delta$ *if* $\overline{\mathbf{s}} \in \partial_\varepsilon^{\emptyset,\emptyset}(s)$, *for all* $s \in \Gamma$, $\overline{\mathbf{s}} \in \Gamma^*$;
- $(q, \varepsilon, s, q, \varepsilon)$, *for all* $s \in \Gamma$ *with* $\mathcal{N}(s)\emptyset$.

Theorem 39 (Automaton correctness). *For all closed expressions $t \in R(\Sigma)$, $\mathcal{L}(t) = \mathcal{L}(\mathcal{UA}(t))$.*

Proof. Let $\mathcal{UA}(t) = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$. We prove a generalized statement from which the original statement follows trivially: for all $\overline{\mathbf{r}} \in \Delta(t)^*$, $\emptyset \vdash w \in \overline{\mathbf{r}}$ iff $(q, \overline{\mathbf{r}}, w) \vdash^* (q, \varepsilon, \varepsilon)$. The proof in the left-to-right direction is by induction on the derivation of $\emptyset \vdash w \in \overline{\mathbf{r}}$.

Case $\emptyset \vdash \varepsilon \in []$. Immediate.

Case $\emptyset \vdash w \in \overline{\mathbf{r}}$ because $w = w_1 w_2$, $\overline{\mathbf{r}} = [r] : \overline{\mathbf{r}}'$, $\emptyset \vdash w_1 \in r$, and $\emptyset \vdash w_2 \in \overline{\mathbf{r}}'$. By induction, we find that $(q, [r], w_1) \vdash^* (q, [], \varepsilon)$. By a standard argument that means $(q, [r] : \overline{\mathbf{r}}', w_1 w_2) \vdash^* (q, \overline{\mathbf{r}}', w_2)$. By the second inductive hypothesis, we find that $(q, \overline{\mathbf{r}}', w_2) \vdash^* (q, [], \varepsilon)$. Taken together, we obtain the desired result.

Now we consider the derivation of $\emptyset \vdash w \in r$ by performing a case analysis on w and using Lemma 27.

Case ε. In this case, $\emptyset \vdash \varepsilon \in r$ iff $\mathcal{N}(r_i)\emptyset$ iff $(q, \varepsilon, r, q, \varepsilon) \in \delta$ so that $(q, [r], \varepsilon) \vdash^+ (q, \varepsilon, \varepsilon)$.

Case aw. In this case $\emptyset \vdash aw \in r$. By Lemma 27, $\emptyset \vdash aw \in r$ is equivalent to $\emptyset \vdash w \in \tilde{\partial}_a^{\emptyset,\emptyset}([r])$ and we perform a subsidiary induction on its definition. That is, either $\exists \overline{\mathbf{s}} \in \partial_a^{\emptyset,\emptyset}(r)$ such that $\emptyset \vdash w \in \overline{\mathbf{s}}$. In that case, $\mathcal{UA}(t)$ has a transition $(q, r, aw) \vdash (q, \overline{\mathbf{s}}, w)$ by definition of δ. By induction we know that $(q, \overline{\mathbf{s}}, w) \vdash^+ (q, \varepsilon, \varepsilon)$.

Alternatively, $\exists \overline{\mathbf{s}} \in \partial_\varepsilon^{\emptyset,\emptyset}(r)$ such that $\emptyset \vdash aw \in \overline{\mathbf{s}}$. In this case, $(q, r, aw) \vdash (q, \overline{\mathbf{s}}, aw)$ is a transition and by induction we have $(q, \overline{\mathbf{s}}, aw) \vdash^+ (q, \varepsilon, \varepsilon)$.

Right-to-left direction. By induction on the length of $(q, \overline{\mathbf{r}}, w) \vdash^* (q, [], \varepsilon)$.

Case length 0: it must be $\overline{\mathbf{r}} = []$ and $w = \varepsilon$. Obviously, $\emptyset \vdash \varepsilon \in []$.

Case length > 0: Thus the first configuration must have the form $(q, [s] : \overline{\mathbf{r}}, w)$. There are three possibilities.

Subcase $(q, [s] : \overline{\mathbf{r}}, w) \vdash (q, \overline{\mathbf{s}} : \overline{\mathbf{r}}, w')$ if $w = aw'$ and $\overline{\mathbf{s}} \in \partial_a(s)$. We split the run of the automaton at the point where $\overline{\mathbf{s}}$ is first consumed: let $w' = w_1 w_2$ such that $(q, \overline{\mathbf{s}} : \overline{\mathbf{r}}, w_1 w_2) \vdash^* (q, \overline{\mathbf{r}}, w_2) \vdash^* (q, [], \varepsilon)$. Hence, there is also a shorter run on w_1: $(q, \overline{\mathbf{s}}, w_1) \vdash^* (q, [], \varepsilon)$. Induction yields $\emptyset \vdash w_1 \in \overline{\mathbf{s}}$. By Lemma 27, we also have a derivation $\emptyset \vdash aw_1 \in s$. By induction on the $\overline{\mathbf{r}}$ run, we obtain $\emptyset \vdash w_2 \in \overline{\mathbf{r}}$ and applying the stack rule yields $\emptyset \vdash aw_1 w_2 \in [s] : \overline{\mathbf{r}}$ or in other words $\emptyset \vdash w \in [s] : \overline{\mathbf{r}}$.

Subcase $(q, [s] : \overline{\mathbf{r}}, w) \vdash (q, \overline{\mathbf{s}} : \overline{\mathbf{r}}, w)$ if $\overline{\mathbf{s}} \in \partial_\varepsilon(s)$. We split the run of the automaton at the point where $\overline{\mathbf{s}}$ is first consumed: let $w' = w_1 w_2$ such that $(q, \overline{\mathbf{s}} : \overline{\mathbf{r}}, w_1 w_2) \vdash^* (q, \overline{\mathbf{r}}, w_2) \vdash^* (q, [], \varepsilon)$. Hence there is also a shorter run on w_1: $(q, \overline{\mathbf{s}}, w_1) \vdash^* (q, [], \varepsilon)$. By induction, we have a derivation $\emptyset \vdash w_1 \in \overline{\mathbf{s}}$, which yields $\emptyset \vdash w_1 \in s$ by Lemma 26, and a derivation $\emptyset \vdash w_2 \in \overline{\mathbf{r}}$, which we can combine to $\emptyset \vdash w_1 w_2 \in [s] : \overline{\mathbf{r}}$ as desired.

Subcase $(q, [s] : \overline{\mathbf{r}}, w) \vdash (q, \overline{\mathbf{r}}, w)$ if $\mathcal{N}(s)\emptyset$. By induction, $\emptyset \vdash w \in \overline{\mathbf{r}}$. As $\mathcal{N}(s)\emptyset$, it must be that $\emptyset \vdash \varepsilon \in s$. Hence, $\emptyset \vdash w \in [s] : \overline{\mathbf{r}}$. □

If all recursion operators in an expression t are guarded, in the sense that they consume some input before entering a recursive call, then all ε-transitions in the constructed automaton pop the stack. In fact, when restricting to guarded expressions, the spontaneous derivative function is not needed at all, which explains the simplicity of the derivative in the work of Winter and coworkers [19].

Acknowledgments. The thoughtful comments of the anonymous reviewers helped improve the presentation of this paper.

References

1. Adams, M.D., Hollenbeck, C., Might, M.: On the complexity and performance of parsing with derivatives. In: PLDI, pp. 224–236. ACM (2016)
2. Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D.: Compilers. Principles, Techniques, and Tools. Addison-Wesley, Boston (2007)
3. Antimirov, V.: Rewriting regular inequalities. In: Reichel, H. (ed.) FCT 1995. LNCS, vol. 965, pp. 116–125. Springer, Heidelberg (1995). doi:10.1007/3-540-60249-6_44
4. Antimirov, V.M.: Partial derivatives of regular expressions and finite automaton constructions. Theor. Comput. Sci. **155**(2), 291–319 (1996)
5. Brzozowski, J.A.: Derivatives of regular expressions. J. ACM **11**(4), 481–494 (1964)
6. Caron, P., Champarnaud, J.-M., Mignot, L.: Partial derivatives of an extended regular expression. In: Dediu, A.-H., Inenaga, S., Martín-Vide, C. (eds.) LATA 2011. LNCS, vol. 6638, pp. 179–191. Springer, Heidelberg (2011). doi:10.1007/978-3-642-21254-3_13
7. Caron, P., Champarnaud, J.-M., Mignot, L.: Multi-tilde-bar derivatives. In: Moreira, N., Reis, R. (eds.) CIAA 2012. LNCS, vol. 7381, pp. 321–328. Springer, Heidelberg (2012). doi:10.1007/978-3-642-31606-7_28
8. Caron, P., Champarnaud, J., Mignot, L.: A general framework for the derivation of regular expressions. RAIRO - Theor. Inf. Appl. **48**(3), 281–305 (2014)
9. Champarnaud, J.-M., Jeanne, H., Mignot, L.: Approximate regular expressions and their derivatives. In: Dediu, A.-H., Martín-Vide, C. (eds.) LATA 2012. LNCS, vol. 7183, pp. 179–191. Springer, Heidelberg (2012). doi:10.1007/978-3-642-28332-1_16
10. Grabmayer, C.: Using proofs by coinduction to find "Traditional" Proofs. In: Fiadeiro, J.L., Harman, N., Roggenbach, M., Rutten, J. (eds.) CALCO 2005. LNCS, vol. 3629, pp. 175–193. Springer, Heidelberg (2005). doi:10.1007/11548133_12

11. Leiß, H.: Towards Kleene algebra with recursion. In: Börger, E., Jäger, G., Kleine Büning, H., Richter, M.M. (eds.) CSL 1991. LNCS, vol. 626, pp. 242–256. Springer, Heidelberg (1992). doi:10.1007/BFb0023771
12. Lombardy, S., Sakarovitch, J.: Derivatives of rational expressions with multiplicity. Theor. Comput. Sci. **332**(1–3), 141–177 (2005)
13. Might, M., Darais, D., Spiewak, D.: Parsing with derivatives: a functional pearl. In: Proceedings of ICFP 2011, pp. 189–195. ACM (2011)
14. Owens, S., Reppy, J., Turon, A.: Regular-expression derivatives reexamined. J. Funct. Program. **19**(2), 173–190 (2009)
15. Roşu, G., Viswanathan, M.: Testing extended regular language membership incrementally by rewriting. In: Nieuwenhuis, R. (ed.) RTA 2003. LNCS, vol. 2706, pp. 499–514. Springer, Heidelberg (2003). doi:10.1007/3-540-44881-0_35
16. Sulzmann, M., Thiemann, P.: Derivatives for regular shuffle expressions. In: Dediu, A.-H., Formenti, E., Martín-Vide, C., Truthe, B. (eds.) LATA 2015. LNCS, vol. 8977, pp. 275–286. Springer, Cham (2015). doi:10.1007/978-3-319-15579-1_21
17. Thiemann, P.: Derivatives for enhanced regular expressions. In: Han, Y.-S., Salomaa, K. (eds.) CIAA 2016. LNCS, vol. 9705, pp. 285–297. Springer, Cham (2016). doi:10.1007/978-3-319-40946-7_24
18. Watson, B.W.: FIRE lite: FAs and REs in C++. In: Raymond, D., Wood, D., Yu, S. (eds.) WIA 1996. LNCS, vol. 1260, pp. 167–188. Springer, Heidelberg (1997). doi:10.1007/3-540-63174-7_14
19. Winter, J., Bonsangue, M.M., Rutten, J.: Context-free languages, coalgebraically. In: Corradini, A., Klin, B., Cîrstea, C. (eds.) CALCO 2011. LNCS, vol. 6859, pp. 359–376. Springer, Heidelberg (2011). doi:10.1007/978-3-642-22944-2_25

Dynamic Complexity of the Dyck Reachability

Patricia Bouyer(✉) and Vincent Jugé(✉)

LSV, CNRS & ENS Cachan, Univ. Paris-Saclay, Cachan, France
{bouyer,juge}@lsv.fr

Abstract. Dynamic complexity is concerned with updating the output of a problem when the input is slightly changed. We study the dynamic complexity of Dyck reachability problems in directed and undirected graphs, where updates may add or delete edges. We show a strong dichotomy between such problems, based on the size of the Dyck alphabet. Some of them are P-complete (under a strong notion of reduction) while the others lie either in DynFO or in NL.

1 Introduction

Dynamic problems and dynamic complexity. In this paper, we focus on the dynamic complexity of some reachability problems. Standard complexity theory aims at developing algorithms that, given an input of some problem, compute an output as efficiently as possible. Its dynamic variant is focused on algorithms that are capable of efficiently updating the output after a small change of the input [11,16,17]. Such algorithms may rely on auxiliary data about the current instance of the problem, and update it when the instance is modified.

A well-studied dynamic complexity class is DynFO. An algorithm is in DynFO if the output and the auxiliary data can be updated by FO formulas after a small change of the input. Variants of DynFO include the class DynFO^+, which allows polynomial-time precomputations, and DynTC^0, in which updates of the auxiliary data are performed by TC^0 circuits.

Consider the problem of reachability in directed graphs, and update operations that consist in inserting or deleting edges (one at a time). It was recently proven that this problem belongs to the class DynFO [3], which had been conjectured for decades.

Furthermore, like static complexity classes, dynamic complexity classes come with natural notions of reduction. The class DynFO is closed under bounded expansion first-order reductions (hereafter called bfo reductions), which are specific L reductions (L is for logarithmic space). A bfo reduction from a problem to another one is a first-order mapping from instances of the first problem to instances of the latter one, such that performing an update operation on the instance of the first problem amounts to performing a bounded number of update operations on the instance of the latter problem. Similarly, the class DynFO^+ is closed under bounded expansion first-order reductions with polynomial-time precomputation (hereafter called bfo^+ reductions).

This work is supported by EU under ERC EQualIS (FP7-308087).

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 265–280, 2017.
DOI: 10.1007/978-3-662-54458-7_16

Reachability problems and language theory. Dyck reachability problems lie at the interface between two areas. On the one hand, language theory is concerned with handling descriptions of languages, that is sets of words, with respect to various questions: Is a language empty, finite or infinite? What about the intersection or the union of two languages? Does a language contain a given word? Among the best known and most simple classes of languages are regular and context-free languages. On the other hand, reachability problems deal with the existence of paths in graphs, and include questions such as: Does there exist a path between two given vertices? How long must be such paths?

Dyck reachability problems are focused on the existence of paths in labeled graphs, whose labels belong to a given *Dyck* language. Dyck languages are languages of well-parenthesized words and, roughly speaking, are the most simple context-free languages that are not regular. The Dyck reachability problem in labeled directed acyclic graphs was proven to be in DynFO [17], when considering two types of update operations on labeled graphs, which are insertion and deletion of edges. Whether this result extends to all labeled directed graphs was then an open question.

Our contributions. We study this open question, and we distinguish the Dyck reachability problem in two different ways. Is the labeled graph directed or undirected? How many symbols does the Dyck alphabet contain?

We prove that there exists a strong dichotomy between the dynamic complexity of such problems, based on the size of the Dyck alphabet. In the case of a unary Dyck alphabet, the Dyck reachability problem lies in NL (nondeterministic logarithmic space), and even lies in DynFO in the case of undirected graphs; this contrasts with the case of binary Dyck alphabets, where we prove that the Dyck reachability problem is P-complete under bfo^+ reductions. Furthermore, it is widely believed [16] that no P-complete problems under bfo^+ reductions lie in classes such as DynFO or the slightly broader class DynFO^+.

Related works. From its very inception 20 years ago, dynamic complexity has been a framework of study for several variants of reachability problems and language theory problems. The class DynFO was shown to contain reachability problems in directed acyclic graphs [5], undirected graphs [16] and, most recently, in all directed graphs [3]; regular and Dyck languages [16], then all context-free languages [6]; Dyck reachability in directed acyclic graphs [17].

At the same time, finding natural problems that are NL- or P-complete (under L reductions) and belong to low dynamic complexity classes such as DynFO, DynFO^+ or DynTC^0 is an ongoing challenge. All known P-complete problems lying in DynFO rely on highly redundant inputs, hence may be seen as artificial [16]. Hence, a notion of *non-redundant projection* [14] was introduced. Non-redundant projections are a special kind of P reductions, which contains, in particular, bfo and bfo^+ reductions.

Hence, for every static complexity class $\mathcal{C}$, we define *non-redundant* $\mathcal{C}$-complete problems as those problems that are $\mathcal{C}$-complete both under L reductions and under non-redundant projections. Most canonical P-complete problems

are non-redundant, hence non-redundancy may be seen as a prerequisite for being a "natural" problem.

A breakthrough was the proof that the Dyck reachability problem in acyclic directed graphs, which is a non-redundant LogCFL-complete problem, belongs to DynFO [17]. It was then proved in [15] that the reachability problem in labeled acyclic graphs, where path labels are constrained to belong to a given context-free grammar (and not only to a Dyck language), is in DynFO. We prove here that the results of [15] are unlikely to extend to all labeled graphs, or even to undirected graphs, even in the simple case of two-letter Dyck languages. This also allows us to answer negatively a question of Weber and Schwentick, who asked in [17] whether "the Dyck reachability problem might be a non-redundant P-complete problem that allows efficient updates."

Complete proofs can be found in research report [2].

2 Definitions

2.1 Dyck Reachability Problems

A labeled directed graph is a triple $G = (V, L, E)$ where V is a finite set of vertices, L is a finite set of labels and $E \subseteq V \times L \times V$ is a finite set of edges. The graph G is said to be *unlabeled* if L is a singleton set; in that case, we may directly represent G as a pair (V, E) where $E \subseteq V \times V$. The graph G is also said to be *undirected* if the relation E is symmetric, i.e. if, for every edge (v, θ, w) in E, the triple (w, θ, v) also belongs to E.

A path in the graph G is a finite sequence of edges $\pi = (v_1, \theta_1, w_1) \cdot (v_2, \theta_2, w_2) \cdot \ldots \cdot (v_k, \theta_k, w_k)$ such that $v_{i+1} = w_i$ for all $i \in \{1, \ldots, k-1\}$. The vertex v_1 is called the *source* of π, and w_k is called the *sink* of π. We also denote by $\lambda(\pi)$ the word $\theta_1 \cdot \ldots \cdot \theta_k$, which is called the *label* of π.

Assume that the label set L is of the form $L = \{\ell_1, \ldots, \ell_n\} \uplus \{\overline{\ell}_1, \ldots, \overline{\ell}_n\}$ for some integer $n \geq 1$. The *Dyck language* (also called *semi-Dyck language* in [7]) associated with L is the context-free language $\mathbf{D}_n$ built over the grammar: $S \to \varepsilon \mid \ell_1 \cdot S \cdot \overline{\ell}_1 \cdot S \mid \ldots \mid \ell_n \cdot S \cdot \overline{\ell}_n \cdot S$, where ε is the empty word. The set $\{\ell_1, \ldots, \ell_n\}$ is said to be the *Dyck alphabet* of that language.

The *n-letter Dyck reachability problem* asks whether, given two vertices s and t of G, there exists a path in G, with source s and sink t, and whose label belongs to the Dyck language $\mathbf{D}_n$ (the actual value of the label set L does not matter, as long as its elements can be partitioned in n ordered pairs). The *n-letter undirected* Dyck reachability problem is the restriction of that problem to the case where the underlying graph G is constrained to be undirected.

2.2 Dynamic Complexity

In this paper, we study the dynamic complexity of Dyck reachability problems. To that end, we first introduce briefly the formalisms of descriptive and dynamic complexity here, and refer to [10,13,16] for more details.

Descriptive complexity aims at characterizing positive instances of a problem using logical formulas. The input is then described as a logical structure described by a set of k-ary predicates (the *vocabulary*) over its universe. For example, a graph can be described by a binary predicate representing its edges, with the set of vertices (usually identified with $\{1, \dots, n\}$ for some n) as the universe. The problem of deciding whether some state has at most one outgoing edge can be described by the first-order formula $\exists x. \forall y. \forall z. (E(x, y) \wedge E(x, z)) \Rightarrow (y = z)$. The class FO contains all problems that can be characterized by such first-order formulas. This class corresponds to the circuit-complexity class AC^0 (under adequate uniformity assumptions) [1].

Dynamic complexity aims at developing algorithms that can efficiently update the output of a problem when the input is slightly changed, for example reachability of one vertex from another one in a graph. We would like our algorithm to take advantage of previous computations in order to very quickly decide the existence of a path in the modified graph.

Formally, a decision problem S is a subset of the set of τ-structures $\mathsf{Struct}(\tau)$ built on a vocabulary τ. In order to turn S into a dynamic problem DynS, we need to define a finite set of allowed updates. For instance, we might use a 2-ary operator $\mathsf{ins}(x, y)$ that would insert an edge between nodes x and y. For a universe of size n, the set of update operations forms a finite alphabet, denoted by Σ_n. A finite word in Σ_n^* then corresponds to a structure obtained by applying a sequence of update operations of Σ_n to the empty structure $\mathcal{I}_n$ over the vocabulary τ. The language DynS_n is defined as the set of those words in Σ_n^* that correspond to structures of S, and DynS is the union (over all n) of all such languages.

A dynamic machine is a uniform family $(M_n)_{n \in \mathbb{N}}$ of deterministic finite automata $M_n = \langle Q_n, \Sigma_n, \delta_n, s_n, F_n \rangle$ over an update alphabet Σ_n, with an update transition function δ_n. Every state is a polynomial-size auxiliary data structure over some vocabulary τ^{aux}, which contains the vocabulary τ. Such a machine solves a dynamic problem if $\mathsf{DynS}_n = \mathcal{L}(M_n)$ for all n. It is in the dynamic complexity class $\mathcal{C}'$-$\mathsf{Dyn}\mathcal{C}$ (or simply $\mathsf{Dyn}\mathcal{C}$ if $\mathcal{C} = \mathcal{C}'$) if the update transition function and membership in the accepting set can be computed in $\mathcal{C}$, while the initial state can be computed in $\mathcal{C}'$. In other words, solving the initial instance of the problem and computing initial auxiliary data structure can be done in $\mathcal{C}'$, and after any update of the input (specified by some letter of Σ_n), further calculations to solve the problem and update the auxiliary data on that new instance are restricted to the class $\mathcal{C}$. Of course, for a dynamic complexity class $\mathcal{C}'$-$\mathsf{Dyn}\mathcal{C}$ to have some interest, the class $\mathcal{C}$ should be easier than the static complexity class of the original problem.

In this paper, we only consider the case where $\mathcal{C} = \mathsf{FO}$, and where $\mathcal{C}' = \mathsf{FO}$ or $\mathcal{C}' = \mathsf{P}$, meaning that first-order formulas will be used to describe how predicates are updated along transitions, and that we may make use of polynomial-time precomputations. As a convention, we will denote the class P-DynFO by DynFO^+, and we recall that the simple notation DynFO is for FO-DynFO.

2.3 Dynamic Reductions

Dynamic complexity comes with the notion of dynamic reductions [16]. Let $\mathcal{C}$ be a complexity class. A (static) $\mathcal{C}$ reduction from a decision problem $\mathcal{P}$ to another decision problem $\mathcal{Q}$ is a mapping in $\mathcal{C}$ from the instances of $\mathcal{P}$ to the instances of $\mathcal{Q}$ that associates every positive instance of $\mathcal{P}$ with a positive instance of $\mathcal{Q}$, and every negative instance of $\mathcal{P}$ with a negative instance of $\mathcal{Q}$. Standard P-completeness results use L reductions [7].

A *dynamic reduction* from a dynamic problem $\mathcal{P}$ (with vocabulary τ_1) to another dynamic problem $\mathcal{Q}$ (with vocabulary τ_2) is a mapping from $\mathsf{Struct}(\tau_1)$ to $\mathsf{Struct}(\tau_2)$ such that:

- every positive (respectively, negative) instance of $\mathcal{P}$ is mapped to a positive (respectively, negative) instance of $\mathcal{Q}$;
- every update on an instance i_1 of $\mathcal{P}$ results in a *well-behaved* sequence of updates on the instance i_2 of $\mathcal{Q}$ to which i_1 is mapped.

Dynamic reductions have therefore several parameters: the complexity class to which the mapping belongs, and the sequences of updates that are allowed.

The dynamic classes DynFO and DynFO^+ are respectively closed under bounded expansion first-order (bfo for short) and bounded expansion first-order with polynomial-time precomputation (bfo^+ for short) reductions [16]. A dynamic reduction μ from $\mathcal{P}$ to $\mathcal{Q}$ is bfo^+ if it is a FO reduction and if every update on an instance i_1 of $\mathcal{P}$ results in a bounded sequence of FO updates on its image $\mu(i_1)$. If, furthermore, the empty structure $\mathcal{I}_1$ is mapped to a structure $\mu(\mathcal{I}_1)$ that can be obtained by applying a bounded sequence of FO updates on the empty structure $\mathcal{I}_2$, then we say that μ is bfo.

Note that dynamic reductions can be applied to the class P (which coincides with the class DynP, under the assumption that updates are one-bit input changes). So, being P-hard for bfo^+ reductions is arguably stronger than being P-hard for L reductions. Furthermore, it is known that the classes of bfo and of bfo^+ reductions are closed under composition and that the circuit value problem is a P-complete problem for bfo^+ reductions [16]. Hence, every P problem to which the circuit value problem is bfo^+-reducible is also P-complete problem for bfo^+ reductions.

2.4 Main Result

We are now in a position to formally present our main result.

Theorem 1. *The* 1*-letter Dyck reachability problem is in* NL*, and the* 1*-letter undirected Dyck reachability problem is in* $\mathsf{NL} \cap \mathsf{DynFO}$*. Furthermore, for all integers* $n \geq 2$*, both the* n*-letter Dyck reachability problem and the* n*-letter undirected Dyck reachability problem are* P*-complete for* bfo^+ *reductions.*

Remark 1. Note that $\mathsf{NL} \cap \mathsf{DynFO}$ is not known to be strictly included in NL. Nevertheless, the case of undirected graph appears to be "easier" than the case of directed graphs in the 1-letter case. Hence, the P-hardness of both cases for alphabets with at least two letters appears rather unexpected.

3 One-Letter (Undirected) Dyck Reachability Problems

We prove here the first part of Theorem 1, that is we assume $n = 1$. We first observe that the 1-letter Dyck reachability problem is equivalent to a standard reachability problem in one-counter automata (without zero-tests), which is known to belong to NL [4,8]. The 1-letter undirected Dyck reachability problem is a restriction of the 1-letter directed Dyck reachability problem, hence it is in NL as well. Furthermore, we make the following claim.

Proposition 1. *Let s and t be two distinct vertices of an undirected labeled graph $G = (V, E, L)$, with $L = \{\ell_1, \overline{\ell}_1\}$. There exists a Dyck path from s to t in G if and only if:*

- *the set $\{x \in V \mid (s, \ell_1, x) \in E\}$ is non-empty;*
- *the set $\{y \in V \mid (t, \overline{\ell}_1, y) \in E\}$ is non-empty;*
- *there exists a path of even length from s to t in G.*

Proof. First, if there exists a Dyck path $\pi = (v_i, \lambda_i, v_{i+1})_{0 \leq i < k}$ with $s = v_0$ and $t = v_k$, then $\lambda_0 = \ell_1$, $\lambda_{k-1} = \overline{\ell}_1$, and the sets $\{0 \leq i < k \mid \lambda_i = \ell_1\}$ and $\{0 \leq i < k \mid \lambda_i = \overline{\ell}_1\}$ have the same cardinality, which proves that k is an even number.

Conversely, assume that $s \neq t$ and that the three conditions of Proposition 1 hold. Let $\pi = (v_i, \lambda_i, v_{i+1})_{0 \leq i < 2k}$ be a path of length $2k$ from s to t in G, for some integer $k \geq 1$. Let κ be the cardinality of the set $\{0 \leq i < 2k \mid \lambda_i = \ell_1\}$ and let $\overline{\kappa}$ be the cardinality of the set $\{0 \leq i < 2k \mid \lambda_i = \overline{\ell}_1\}$. Since $\kappa + \overline{\kappa} = 2k$, we have $\overline{\kappa} - \kappa = 2(k - \kappa)$.

Furthermore, consider vertices $x, y \in V$ such that (s, ℓ_1, x) and $(t, \overline{\ell}_1, y)$ belong to E. Since the graph is undirected, there exist also edges (x, ℓ_1, s) and $(y, \overline{\ell}_1, t)$. Let ρ_1 be the length-2 circuit $(s, \ell_1, x) \cdot (x, \ell_1, s)$, and let ρ_2 be the length-2 circuit $(t, \overline{\ell}_1, y) \cdot (y, \overline{\ell}_1, t)$. One checks easily that the path $\rho_1^k \cdot \pi \cdot \rho_2^\kappa$ is a Dyck path in G, where ρ_1^k is the concatenation of k occurrences of ρ_1, and ρ_2^κ is the concatenation of κ occurrences of ρ_2. □

Hence, checking whether there exists a Dyck path from s to t in G amounts to checking whether $s = t$ or, if $s \neq t$, whether the sets $\{x \in V \mid (s, \ell_1, x) \in E\}$ and $\{y \in V \mid (t, \overline{\ell}_1, y) \in E\}$ are non-empty, and whether there exists a path of even length from s to t in G. The first statements can be checked directly in FO, and the latter one can be checked in DynFO, as proved below. This completes the proof of the first part of Theorem 1 in the case $n = 1$.

Lemma 1. *Checking whether there exists a path of even length from s to t in G is feasible in* DynFO.

Proof. Let Γ be the graph $(G \times \{0, 1\}, E')$, where $E' = \{((x, 0), \ell, (y, 1)) \mid (x, \ell, y) \in G\} \cup \{((x, 1), \ell, (y, 0)) \mid (x, \ell, y) \in G\}$. The graph Γ consists in two copies of G, and edges of G translate into edges between these two copies. Since Γ is undirected, the reachability problem in Γ is in DynFO [3,16]. Furthermore, there exists a path of even length from s to t in G if and only if there exists

a path from $(s,0)$ to $(t,0)$ in Γ. Since Γ is FO-definable in terms of G, and since adding/deleting one edge in G amounts to adding/deleting two edges in Γ, Lemma 1 follows. □

Remark 2. Note that this proof heavily relies on the property that the graph is undirected. In fact, the problem of computing distances in directed graphs, whose membership in DynFO or DynFO^+ is a long-standing open question [3,9], is bfo^+-reducible to the 1-letter Dyck reachability problem (over directed graphs). The reduction is as follows.

Given an unlabeled directed graph $G = (V, E)$, equip each edge with a label ℓ_1, and add self-loops (with the label ℓ_1) around each vertex in V. Then, for all vertices $v \in V$, add n vertices $(v,1), \ldots, (v,n)$, where $n = |V|$, and add edges with the label $\overline{\ell}_1$ from v to $(v,1)$ and from (v,i) to $(v,i+1)$, for all i. It comes at once that the distance (in the original graph G) from a vertex s to a vertex t is k if and only if there exists a Dyck path (in the extended, labeled graph) from s to (t,k) but not to $(t,k-1)$.

Furthermore, the proof of [9] showing that distances in graphs can be computed in DynTC^0 does not extend to the 1-letter Dyck reachability, whose precise dynamic complexity remains therefore unknown.

4 *n*-letter Dyck Reachability Problem

We prove now that, for all integers $n \geq 2$, the n-letter Dyck reachability problem is P-complete for bfo^+ reductions.

We first introduce two auxiliary problems.

1. Let $G = (V, E)$ be an unlabeled directed graph, let $(V_\wedge, V_\vee)$ be a partition of V, and let s and t be two marked vertices of G. The *alternating reachability problem* asks whether s belongs to the *alternating coaccessible set* of t, i.e. the smallest subset X of V such that all of $\{t\}$, $\{x \in V_\vee \mid \exists y \in X \text{ s.t. } (x,y) \in E\}$ and $\{x \in V_\wedge \mid \forall y \in V, (x,y) \in E \Rightarrow y \in X\}$ are subsets of X.
   Note that this problem could be alternatively and equivalently defined using the notion of winning state in a two-player turn-based zero-sum reachability game. However, we choose the above definition using a fixed point to avoid defining the notion of winning strategies.
2. Let $G = (V, E, L)$ be a labeled directed graph with set of labels $L = V \cup \{\overline{v} \mid v \in V\}$, and let s and t be two marked vertices of G. A *near-Dyck word* is an element of the set $\mathbf{D'}$ built over the grammar: $S \to \varepsilon \mid v \cdot S \cdot \overline{v} \cdot S$ (for all $v \in V$). The *near-Dyck reachability problem* asks whether there exists a path π in G, with source s, sink t, and whose label belongs to $\mathbf{D'}$.
   Note that the near-Dyck reachability problem may be viewed as a generalisation of the n-letter Dyck reachability problem: it involves a grammar with $|V|$ rules and not only n, i.e. the size of the grammar is not constant anymore.

While it is well-known that the alternating reachability problem is P-hard for standard logarithmic-space reductions, it is also the case that it is P-hard

for bfo^+ reductions [16]. Hence, we show in the two next subsections that there exists a bfo^+ reduction from the alternating reachability problem to the near-Dyck reachability problem, and that there exists a bfo^+ reduction from that latter problem to the 2-letter Dyck reachability problem. It will follow that the 2-letter (and, therefore, the n-letter) Dyck reachability problem is P-hard for bfo^+ reductions.

On the other hand, it is known that the n-letter Dyck reachability problem belongs to P (see [7, Sect. A.7.9]).

4.1 From the Near-Dyck Reachability Problem to the Dyck Reachability Problem

Let $G = (V, E, L)$ be a labeled directed graph with set of labels $L = V \cup \overline{V}$ (where $\overline{V} = \{\overline{v} \mid v \in V\}$), and let s and t be two marked vertices of G.

We fix the new alphabet $\mathcal{L} = \{0, 1, \overline{0}, \overline{1}\}$. Then, we consider an integer n and an injective *coding* function $\mathsf{cod} : (V \cup \overline{V}) \mapsto \mathcal{L}^n$ such that $\mathsf{cod}(V) \subseteq \{0,1\}^n$, $\mathsf{cod}(\overline{V}) \subseteq \{\overline{0}, \overline{1}\}^n$. We further assume the following consistency requirement about cod: for all $v \in V$ and all $i \in \{1, \dots, n\}$, we have $\mathsf{cod}(\overline{v})_i = \overline{\mathsf{cod}(v)_{n+1-i}}$, where we denote by w_i the i^{th} letter of the word $w \in \mathcal{L}^n$.

Formally, let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{L})$ be the labeled directed graph defined by:

- $\mathcal{V} = V \cup \big(V \times (V \cup \overline{V}) \times \{0, 1, \dots, n\}\big)$;
- $\mathcal{L} = \{0, 1, \overline{0}, \overline{1}\}$;
- $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2$, where

$$\mathcal{E}_1 = \{x \xrightarrow{0} (x, v, 0) \mid x, v \in V\} \cup \{x \xrightarrow{\overline{0}} (x, \overline{v}, 0) \mid x \in V, \overline{v} \in \overline{V}\} \cup \{(x, v, i) \xrightarrow{\mathsf{cod}(v)_{i+1}} (x, v, i+1) \mid x \in V, v \in V \cup \overline{V}, 0 \leq i \leq n-1\} \text{ and}$$

$$\mathcal{E}_2 = \{(x, v, n) \xrightarrow{0} y \mid x \xrightarrow{v} y \in E\} \cup \{(x, \overline{v}, n) \xrightarrow{\overline{0}} y \mid x \xrightarrow{\overline{v}} y \in E\},$$

and in which we mark the vertices s and t.

Each sequence of transitions $x \xrightarrow{0} (x, v, 0) \xrightarrow{v_1} \dots \xrightarrow{v_n} (x, v, n)$ prepares the encoding of some edge leaving x with label v. If there is some edge $x \xrightarrow{v} y$ in the original graph, then only one edge $(x, v, n) \xrightarrow{0} y$ needs to be added: this is the role of the edges in $\mathcal{E}_2$. We use a similar encoding for edges labeled by $\overline{v}$.

Proposition 2. *There exists a near-Dyck path from s to t in G if and only if there exists a Dyck path from s to t in $\mathcal{G}$.*

Proof. First, for every pair $(u, v) \in \mathcal{V}^2$, there exists at most one edge in $\mathcal{E}$ with source u and sink v. Henceforth, we omit representing labels of edges and of paths in $\mathcal{G}$.

We further define two mappings φ and ψ. The mapping φ identifies every label $\lambda \in L$ with a word $\varphi(\lambda) \in \mathcal{L}^*$, as follows:

$$\varphi(v) = 0 \cdot \mathsf{cod}(v) \cdot 0 \text{ for all } v \in V, \text{ and } \varphi(\overline{v}) = \overline{0} \cdot \mathsf{cod}(\overline{v}) \cdot \overline{0} \text{ for all } \overline{v} \in \overline{V}.$$

This mapping extends immediately to a morphism from L^* to $\mathcal{L}^*$ that maps every near-Dyck word $w \in \mathbf{D}'$ to a Dyck word $\varphi(w) \in \mathbf{D}$. The mapping ψ identifies every edge $e \in E$ with a path $\psi(e)$ in $\mathcal{G}$, as follows:

$$\psi(x \xrightarrow{v} y) = (x \to (x, v, 0) \to \ldots \to (x, v, n) \to y) \text{ for all } v \in V \cup \overline{V}.$$

This mapping extends immediately to a morphism that maps every path in G to a path in $\mathcal{G}$. The relation $\lambda(\psi(e)) = \varphi(\lambda(e))$ holds for all edges $e \in E$, and therefore extends to all paths π in G. Hence, a path π in G is near-Dyck if and only if the path $\psi(\pi)$ in $\mathcal{G}$ is Dyck.

In addition, let us call *nominal* paths in $\mathcal{G}$ the paths that belong to the set $\{\psi(e) \mid e \in E\}$, and *generic* paths in $\mathcal{G}$ the concatenations of nominal paths. Nominal paths are the minimal paths whose source and sink both belong to the subset V of $\mathcal{V}$. Hence, every path π from s to t in $\mathcal{G}$ is generic, thus π is the image by ψ of some path $\psi^{-1}(\pi)$ from s to t in G. □

The graph $\mathcal{G}$ is FO-definable as a function of G and of the coding function cod, and adding/deleting an edge in E amounts to adding/deleting exactly one edge in $\mathcal{E}_2$. Since the function cod can be precomputed in P, and due to Proposition 2, the near-Dyck reachability problem is therefore bfo^+-reducible to the Dyck reachability problem.

4.2 From the Alternating Reachability Problem to the Near-Dyck Reachability Problem

Let $G = (V, E)$ be an unlabeled directed graph, let $(V_\wedge, V_\vee)$ be a partition of V, let s and t be two marked vertices of G.

Let us number the vertices of G from 0 to $n-1$, i.e. set $V = \{v_0, \ldots, v_{n-1}\}$. Then, let $\mathbf{G}$ be the context-free grammar with set of non-terminal symbols V and initial symbol s, without terminal symbol, and that consists in three kinds of rules:

- a termination rule $t \to \varepsilon$;
- rules $v \to w$ for all vertices $v \in V_\vee$ and $w \in V$ such that $(v, w) \in E$;
- rules $v \to w_0 \cdot w_1 \cdots w_{n-1}$ for all vertices $v \in V_\wedge$, where $w_i = v_i$ if $(v, v_i) \in E$ and $w_i = t$ otherwise.

The following result is straightforward.

Proposition 3. *Let X be the alternating coaccessible set of t. The vertex s belongs to X if and only if the language generated by $\mathbf{G}$ is non-empty.*

Inspired by the translation of context-free grammars into pushdown automata (by simulating leftmost derivations, see for instance [12, Theorem 6.13]), we build below a labeled graph whose labels correspond to push and pop moves of such a pushdown automaton, so that near-Dyck paths in the new graph will correspond to the empty-stack accepting runs of the pushdown automaton.

Formally, let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{L})$ be the labeled directed graph defined by:

- $\mathcal{V} = \{\circ, \bullet\} \cup V \cup \big(V_\wedge \times \{1, \dots, n-1\}\big)$, where $\circ$ and $\bullet$ are two fresh vertex symbols;
- $\mathcal{L} = V \cup \{\overline{v} \mid v \in V\}$;
- $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2$, where

$$\mathcal{E}_1 = \{\circ \xrightarrow{s} \bullet\} \cup \{\bullet \xrightarrow{\overline{x}} x \mid x \in V\} \cup \{\bullet \xrightarrow{\overline{t}} \bullet\} \text{ and}$$

$$\mathcal{E}_2 = \{x \xrightarrow{y} \bullet \mid x \in V_\vee \text{ and } (x,y) \in E\} \cup$$
$$\{(x, n-1-i) \xrightarrow{v_i} (x, n-i) \mid x \in V_\wedge, 0 \le i \le n-1 \text{ and } (x, v_i) \in E\} \cup$$
$$\{(x, n-1-i) \xrightarrow{t} (x, n-i) \mid x \in V_\wedge, 0 \le i \le n-1 \text{ and } (x, v_i) \notin E\},$$

where we use $(v, 0)$ as a placeholder for v and (v, n) as a placeholder for $\bullet$ (for all vertices $v \in V_\wedge$). Then, let us mark vertices $\circ$ and $\bullet$.

The construction is illustrated in Fig. 1, in which the graph G is associated with the grammar $\mathbf{G}$ whose initial symbol is v_0 and whose rules are $v_3 \to \varepsilon \mid v_1 \to v_2 \mid v_1 \to v_4 \mid v_2 \to v_3 \mid v_3 \to v_1 \mid v_0 \to v_3 v_1 v_3 v_3 v_4 \mid v_4 \to v_3 v_1 v_3 v_3 v_3$.

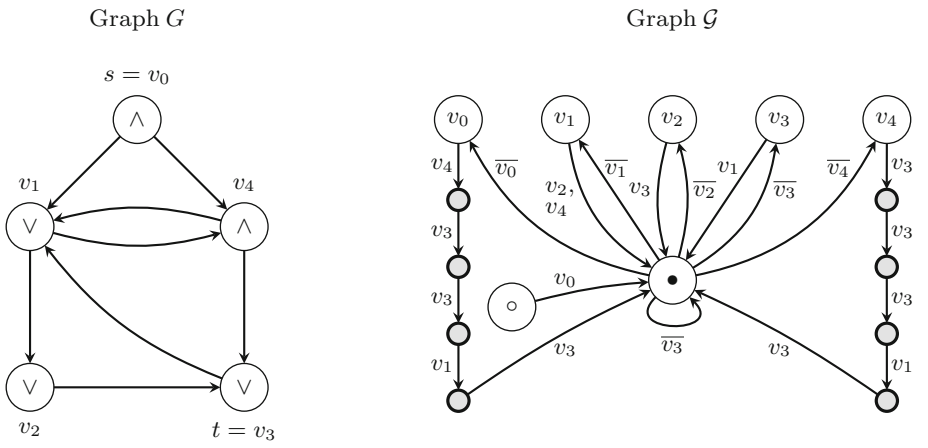


Fig. 1. Graphs G and $\mathcal{G}$

Proposition 4. *There exists a near-Dyck path from $\circ$ to $\bullet$ in $\mathcal{G}$ if and only if the language of the context-free grammar $\mathbf{G}$ is non-empty.*

Proof. Consider the pushdown automaton $\mathcal{A} = (Q, \Sigma, \delta, \iota, F)$, with set of states $Q = \mathcal{V}$, input alphabet $\Sigma = \emptyset$, initial state $\iota = \circ$, set of final states $F = \{\bullet\}$, whose transition function δ contains only the following ε-transitions:

$$\delta : q \xrightarrow{\text{push}(v)} q' \text{ for all } q, q' \in \mathcal{V} \text{ and } v \in V \text{ s.t. } (q, v, q') \in \mathcal{E}$$
$$q \xrightarrow{\text{pop}(v)} q' \text{ for all } q, q' \in \mathcal{V} \text{ and } v \in V \text{ s.t. } (q, \overline{v}, q') \in \mathcal{E},$$

and whose accepting runs are those that start in ι with an empty tape and end in a final state (i.e. in $\bullet$) with an empty tape.

It is straightforward that $\mathcal{A}$ accepts the language of $\mathbf{G}$, for instance by observing that it follows the construction of [12, Theorem 6.13]. Furthermore, near-Dyck paths from $\circ$ to $\bullet$ in $\mathcal{G}$ can be identified with stack operations of accepting executions of $\mathcal{A}$. Proposition 4 follows. □

The graph $\mathcal{G}$ is FO-definable as a function of G, s, t and of the mapping $i \mapsto v_i$. Moreover, adding/deleting an edge e in E amounts to adding/deleting exactly either one or two edges in $\mathcal{E}_2$. Since the mapping $i \mapsto v_i$ can be precomputed in P, and due to Propositions 3 and 4, the alternating reachability problem is therefore bfo^+-reducible to the near-Dyck reachability problem.

5 n-letter Undirected Dyck Reachability Problem

We proceed by proving that there exists a bfo^+ reduction from the 2-letter Dyck reachability problem (in directed graphs) to the 2-letter undirected reachability problem.

Let $G = (V, E, L)$ be a directed labeled graph, with $L = \{\ell_1, \ell_2, \overline{\ell}_1, \overline{\ell}_2\}$, and let s and t be two marked nodes of G. In addition, let $\mathcal{L} = \{0, 1, \overline{0}, \overline{1}\}$ be another set of labels.

The main difficulty, when working in an undirected graph, is that lots of cycles are created, generating lots of stuttering in the words labeling the paths. It is therefore hard to really control where a path goes just by looking at its label. In particular, the recipe used in Sect. 4.1 to reduce the near-Dyck reachability problem to the 2-letter Dyck reachability problem cannot be used now, and it is not clear whether simple alternative reductions from the near-Dyck undirected reachability problem to the 2-letter undirected Dyck reachability problem exist. Below, we prove directly the P-hardness of the 2-letter undirected Dyck reachability. In order to do so, we rely on a rather intricate encoding, where each part plays an important role.

We denote by $\varphi : L^* \mapsto \mathcal{L}^*$ the homomorphism of monoids defined by:

$$\varphi(\ell_1) = 0 \cdot \boxed{\overline{0} \cdot 1} \cdot 1 \cdot 0 \cdot 0 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot \boxed{\overline{1} \cdot 0} \qquad \varphi(\overline{\ell}_1) = \boxed{\overline{0} \cdot 1} \cdot \overline{1} \cdot \overline{1} \cdot \overline{1} \cdot \overline{1} \cdot \overline{0} \cdot \overline{0} \cdot \overline{1} \cdot \boxed{\overline{1} \cdot 0} \cdot \overline{0}$$

$$\varphi(\ell_2) = 0 \cdot \boxed{\overline{0} \cdot 1} \cdot 0 \cdot 0 \cdot 1 \cdot 1 \cdot 0 \cdot 0 \cdot 1 \cdot \boxed{\overline{1} \cdot 0} \qquad \varphi(\overline{\ell}_2) = \boxed{\overline{0} \cdot 1} \cdot \overline{1} \cdot \overline{0} \cdot \overline{0} \cdot \overline{1} \cdot \overline{1} \cdot \overline{0} \cdot \overline{0} \cdot \boxed{\overline{1} \cdot 0} \cdot \overline{0}$$

Observe that the words $\varphi(\overline{\ell}_1)$ and $\varphi(\overline{\ell}_2)$ are formal inverses of the words $\varphi(\ell_1)$ and $\varphi(\ell_2)$: in particular, both the words $\varphi(\ell_1) \cdot \varphi(\overline{\ell}_1)$ and $\varphi(\ell_2) \cdot \varphi(\overline{\ell}_2)$ are Dyck words.

In gray boxes are locks: *along a Dyck path*, once a lock has been traveled through, we cannot go back earlier in the encoding, since this would create a factor $1 \cdot \overline{0}$ or $0 \cdot \overline{1}$, which is not a factor of any Dyck word. We therefore say that a path is *doomed* if it crosses a lock backwards, thereby having a factor $1 \cdot \overline{0}$ or $0 \cdot \overline{1}$. By preventing Dyck paths from having doomed subpaths, locks will allow us to recover partially the directed character of G.

Finally, for every word $w \in \mathcal{L}^*$, we denote by w_i the i^{th} letter of w. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{L})$ be the undirected labeled graph defined by:

- $\mathcal{V} = V \cup (V \times L \times V \times \{1, \dots, 11\})$;
- $\mathcal{E} = \mathcal{E}_{\mathsf{init}} \cup \mathcal{E}_{\mathsf{mid}} \cup \mathcal{E}_{\mathsf{end}}$, where

$$\begin{aligned}
\mathcal{E}_{\mathsf{init}} &= \{x \xleftrightarrow{\varphi(\lambda)_1} (x, \lambda, y, 1) \mid (x, \lambda, y) \in E\} \\
\mathcal{E}_{\mathsf{mid}} &= \{(x, \lambda, y, i-1) \xleftrightarrow{\varphi(\lambda)_i} (x, \lambda, y, i) \mid (x, \lambda, y) \in E, 1 \leq i \leq 11\} \\
\mathcal{E}_{\mathsf{end}} &= \{(x, \lambda, y, 11) \xleftrightarrow{\varphi(\lambda)_{12}} y \mid (x, \lambda, y) \in E\},
\end{aligned}$$

and in which we mark the vertices s and t.

Like in Sects. 4.1 and 4.2, we observe an equivalence between the two kinds of Dyck reachability problems in the graphs G and $\mathcal{G}$, which we prove formally in the rest of the section.

Proposition 5. *There exists a Dyck path from s to t in G if and only if there exists a Dyck path from s to t in $\mathcal{G}$.*

A first *completeness* result towards proving Proposition 5 comes quickly.

Lemma 2. *Let ρ be a Dyck path from s to t in G. There exists a Dyck path from s to t in $\mathcal{G}$.*

Proof. First, for every pair $(u, v) \in \mathcal{V}^2$ there exists at most one undirected edge between u and v in $\mathcal{E}$. Henceforth, we may omit representing labels of edges and of paths in $\mathcal{G}$.

Now, let us denote by ψ the mapping that identifies every edge $e \in E$ with the path $\psi(e) = (x \to (x, \theta, y, 1) \to \dots \to (x, \theta, y, 11) \to y)$ in $\mathcal{G}$, where $e = (x, \theta, y)$. Observe that ψ extends immediately to a morphism that maps every path in G to a path in $\mathcal{G}$. The relation $\lambda(\psi(e)) = \varphi(\lambda(e))$ holds for all edges $e \in E$, and therefore extends to all paths in G. Hence, a path ρ in G is Dyck if and only if the path $\psi(\rho)$ in $\mathcal{G}$ is Dyck. □

However, unlike in Sect. 4.1, there may exist Dyck paths in $\mathcal{G}$ that are not of the form $\psi(\rho)$, as shown by the examples of the two Dyck cycles γ_1 and γ_2 displayed in Fig. 2. Consequently, we cannot use directly the morphism ψ to associate every Dyck path in $\mathcal{G}$ with a Dyck path in G.

We overcome this problem as follows. Let $\mathcal{Q}$ be the set of all factors of all Dyck words with letters in $\mathcal{L}$ (called *approximate Dyck words*), and let $\mathcal{P}$ the set of all paths π in $\mathcal{G}$ such that $\lambda(\pi) \in \mathcal{Q}$ (called *approximate Dyck paths*). Moreover, for every set S of paths, we denote by $\lambda(S)$ the set of labels of paths in S, i.e. $\lambda(S) = \{\lambda(\pi) \mid \pi \in S\}$. It comes at once that $\lambda(\mathcal{P}) \subseteq \mathcal{Q}$, that $\mathcal{Q}$ is factor closed, and that none of the words $1 \cdot \overline{0}$ nor $0 \cdot \overline{1}$ belongs to $\mathcal{Q}$.

We further say that a path in $\mathcal{G}$ is *nominal* if its source and sink belong to V, while its intermediate vertices belong to $\mathcal{V} \setminus V$. For all edges $(x, \lambda, y) \in E$, we denote by $\mathcal{P}_{x,\lambda,y}$ the set of nominal paths $\pi \in \mathcal{P}$ such that π has source x,

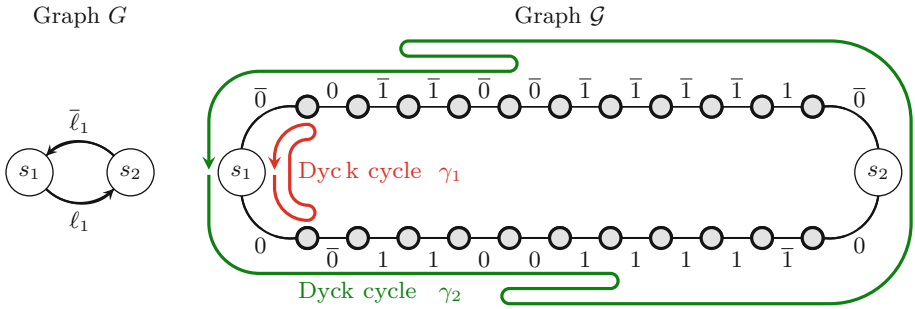


Fig. 2. Graphs G and $\mathcal{G}$, and Dyck cycle in $\mathcal{G}$

sink y, and such that its internal vertices are exactly the elements of the set $\{(x, \lambda, y, i) \mid 1 \leq i \leq 11\}$. For all vertices $x \in V$, we also denote by $\mathcal{P}_x$ the set of nominal paths $\pi \in \mathcal{P}$ such that π has source and sink x, and whose edges are all labeled with 0 or $\overline{0}$. These two classes of paths capture the entire family of nominal paths that belong to $\mathcal{P}$, as shown by the following result.

Lemma 3. *Let $\pi \in \mathcal{P}$ be a nominal path in $\mathcal{G}$. Either there exists an edge $(x, \lambda, y) \in E$ such that $\pi \in \mathcal{P}_{x,\lambda,y}$ or there exists a vertex $x \in V$ such that $\pi \in \mathcal{P}_x$. Moreover, the sets $\mathcal{P}_{x,\lambda,y}$ and $\mathcal{P}_x$ are pairwise disjoint.*

Proof. We first assume that some edge e in π is labeled by 1 or $\overline{1}$. By construction, there exists a unique edge $(x, \lambda, y) \in E$ and a unique pair of integers $i, j \in \{1, \ldots, 11\}$ such that $e = (x, \lambda, y, i) \to (x, \lambda, y, j)$, with $j = i \pm 1$. Since π is nominal, its internal vertices belong to the set $\{(x, \lambda, y, i) \mid 1 \leq i \leq 11\}$, and its source and sink belong to $\{x, y\}$. Then, since π belongs to $\mathcal{P}$, it does not contain doomed paths, hence its source must be x and its sink must be y.

Then, assume that no edge in π is labeled by 1 or $\overline{1}$. Since π is nominal, there exists a unique edge $(x, \lambda, y) \in E$ such that the internal vertices of π belong to the set $\{(x, \lambda, y, i) \mid 1 \leq i \leq 11\}$, and its source and sink belong to $\{x, y\}$. If x is the source of π, then π can never reach the vertex $(x, \lambda, y, 3)$, hence x is the sink of π; if y is the source of π, then π can never reach the vertex $(x, \lambda, y, 9)$, hence y is the sink of π.

Observing that every path in every set $\mathcal{P}_{x,\lambda,y}$ contains an edge labeled by 1 or $\overline{1}$ completes the proof. □

Going further, we associate with every path $\rho = (v_1, \lambda_1, w_1) \cdot \ldots \cdot (v_k, \lambda_k, w_k)$ in G the set $\overline{\mathcal{P}}_\rho$ of paths in $\mathcal{G}$ defined by:

$$\overline{\mathcal{P}}_\rho = \mathcal{P}^*_{v_1} \cdot \mathcal{P}_{v_1,\lambda_1,w_1} \cdot \mathcal{P}^*_{v_2} \cdot \mathcal{P}_{v_2,\lambda_2,w_2} \cdot \ldots \cdot \mathcal{P}^*_{v_k} \cdot \mathcal{P}_{v_k,\lambda_k,w_k} \cdot \mathcal{P}^*_{w_k}.$$

Observe that, unlike the sets $\mathcal{P}_x$ and $\mathcal{P}_{x,\lambda,y}$, the sets $\overline{\mathcal{P}}_\rho$ may contain paths that are not nominal and/or not approximate Dyck paths.

Conversely, however, it comes immediately that every Dyck path π in $\mathcal{G}$ belongs to one unique set $\overline{\mathcal{P}}_\rho$, where ρ is the *nominal ancestor* of π defined below.

Definition 1. *Let π be a Dyck path in $\mathcal{G}$ from s to t. There exists a unique sequence of vertices $v_0, \dots, v_k$, a unique partial function $f_\pi : \{1, \dots, k\} \mapsto L$, whose domain is denoted by $\mathsf{dom}(f_\pi)$, and a unique sequence of nominal paths $\pi_1, \dots, \pi_k$ such that:*

- *$v_0 = s$ and $v_k = t$;*
- *for all $i \in \mathsf{dom}(f_\pi)$, the edge $(v_{i-1}, f_\pi(i), v_i)$ belongs to E, and $\pi_i \in \mathcal{P}_{v_{i-1}, f_\pi(i), v_i}$;*
- *for all $i \in \{1, \dots, k\} \setminus \mathsf{dom}(f_\pi)$, we have $v_{i-1} = v_i$, and $\pi_i \in \mathcal{P}_{v_i}$;*
- *$\pi = \pi_1 \cdot \ldots \cdot \pi_k$.*

We call nominal vertex sequence *of π sequence $v_0, \dots, v_k$,* nominal label mapping *of π the mapping f_π,* nominal decomposition *of π the sequence $\pi_1, \dots, \pi_k$, and* nominal ancestor *of π the path $(v_{i-1}, f_\pi(i), v_i)_{i \in \mathsf{dom}(f_\pi)}$.*

Associating every Dyck path in $\mathcal{G}$ with a unique path in G is a first step towards proving the soundness of the construction. Further steps depend on the following, additional properties of the encoding.

Every Dyck path traveling through the word $\varphi(\ell_1)$, may go back and forth arbitrarily, except at locks, which it may cross only once. Consider such a possible journey through the word $\varphi(\ell_1)$, and observe the word w obtained during that journey. This word is made of blocks that consist, alternatively, of letters 0 and $\overline{0}$, and of letters 1 and $\overline{1}$. Such a word, *even if we first reduce it* (by deleting recursively the words $0 \cdot \overline{0}$ and $1 \cdot \overline{1}$), will always satisfy the following properties:

1. there exists at least four such (non-empty) blocks;
2. the last block consists of letters 0 only;
3. the last two blocks are of odd length, and every other block is of even length.

To illustrate the above analysis, consider the direct journey through $\varphi(\ell_1)$, where we have identified the blocks: $(0 \cdot \overline{0}) \cdot (1 \cdot 1) \cdot (0 \cdot 0) \cdot (1 \cdot 1 \cdot 1 \cdot 1 \cdot \overline{1}) \cdot (0)$. If that word is reduced, there remain four non-empty blocks: $(1 \cdot 1) \cdot (0 \cdot 0) \cdot (1 \cdot 1 \cdot 1) \cdot (0)$.

Another example is the journey first followed by the path γ_2 (see Fig. 2) from the vertex s_1 to the vertex s_2, and which gives us more blocks: $(0 \cdot \overline{0}) \cdot (1 \cdot 1) \cdot (0 \cdot 0) \cdot (1 \cdot 1) \cdot (0 \cdot 0) \cdot (1 \cdot 1 \cdot 1 \cdot 1 \cdot \overline{1}) \cdot (0)$. If that word is reduced, there remain six non-empty blocks: $(1 \cdot 1) \cdot (0 \cdot 0) \cdot (1 \cdot 1) \cdot (0 \cdot 0) \cdot (1 \cdot 1 \cdot 1) \cdot (0)$. One checks easily on these examples that all the words obtained satisfy the properties 1–3.

Properties 1–2 hold for the word $\varphi(\ell_2)$, while property 3 should be replaced by:

3'. the first block of letters 1 and $\overline{1}$ and the last block of letters 0 are of odd length, and every other block is of even length.

This distinction between encodings of the two letters will allow identifying a path that encodes ℓ_1 or ℓ_2, even when there is backtracking between the two locks.

Now, we denote by $\mathcal{Q}_{\mathsf{init}}$ the set of all prefixes of all Dyck words with letters in $\mathcal{L}$. Observe that, for all words $\rho_1, \rho_2 \in \mathcal{L}^*$, the three words $\rho_1 \cdot \rho_2$, $\rho_1 \cdot 0 \cdot \overline{0} \cdot \rho_2$ and $\rho_1 \cdot 1 \cdot \overline{1} \cdot \rho_2$ are either all Dyck or all non-Dyck.

Then, for every word $w \in \mathcal{L}^*$, we call *reduced word* of w the word $\mathsf{red}(w)$ obtained from w by deleting recursively the 2-letter words $0 \cdot \overline{0}$ or $1 \cdot \overline{1}$. Alternatively, if we consider w as an element of the free group generated by ℓ_1 and ℓ_2 (with inverses $\overline{\ell}_1$ and $\overline{\ell}_2$), then $\mathsf{red}(w)$ is the reduced word representing w. We just proved that w is Dyck if and only if $\mathsf{red}(w)$ is Dyck. Moreover, it comes immediately that $\mathcal{Q}_{\mathsf{init}}$ is in fact the set of all words w such that $\mathsf{red}(w)$ has only letters 0 and 1.

The above remarks and notions lead to the following results, whose proofs are then technical yet simple, and therefore omitted here.

Lemma 4. *Let ρ be a path in G. If ρ is not an approximate Dyck path, then $\lambda(\overline{\mathcal{P}}_\rho) \cap \mathcal{Q} = \emptyset$, and if $\lambda(\rho)$ is not a prefix of a Dyck word, then $\lambda(\overline{\mathcal{P}}_\rho) \cap \mathcal{Q}_{\mathsf{init}} = \emptyset$.*

Lemma 5. *Let π be a Dyck path from s to t in $\mathcal{G}$, let ρ be the nominal ancestor of π, and let $\lambda(\rho) \in L^*$ be the label of ρ. In addition, let $\mu : L^* \mapsto \mathbb{Z}$ be the morphism of monoids defined by $\mu(\ell_1) = \mu(\ell_2) = 1$ and $\mu(\overline{\ell}_1) = \mu(\overline{\ell}_2) = -1$. Then, we have $\mu(\lambda(\rho)) = 0$.*

A consequence of Lemmas 4 and 5 is the *correctness* of the construction, which is therefore valid.

Proof (Proposition 5). First, if there exists a Dyck path from s to t in G, then Lemma 2 already states that there also exists a Dyck path from s to t in $\mathcal{G}$. Hence, we look at the converse implication.

Let π be a Dyck path from s to t in $\mathcal{G}$, and let ρ be the nominal ancestor of π. Let $\lambda(\rho)$ be the label of ρ and let Λ be the reduction of $\lambda(\rho)$. Lemma 4 proves that $\lambda(\rho)$ is a prefix of a Dyck word, hence that Λ has only letters ℓ_1 and ℓ_2. Since Lemma 5 also proves that $\mu(\lambda(\rho)) = \mu(\Lambda) = 0$, it follows that Λ is the empty word, i.e. that $\lambda(\rho)$ is a Dyck word. □

We complete the proof of Theorem 1 as follows. Observe that the graph $\mathcal{G}$ is FO-definable as a function of G. Furthermore, adding/deleting an edge in E amounts to adding/deleting exactly twelve edges in $\mathcal{E}$. Due to Proposition 5, the 2-letter Dyck reachability problem is therefore bfo-reducible to the 2-letter undirected Dyck reachability problem.

On the other hand, as a restriction of the n-letter Dyck reachability problem, the n-letter undirected Dyck reachability problem is clearly in P, which completes the proof of Theorem 1.

References

1. Mix Barrington, D.A., Immerman, N., Straubing, H.: On uniformity within NC^1. J. Comput. Syst. Sci. **41**(3), 274–306 (1990)
2. Bouyer, P., Jugé, V.: Dynamic complexity of the Dyck reachability. Research Report arXiv/1610.07499, Computing Research Repository, October 2016
3. Datta, S., Kulkarni, R., Mukherjee, A., Schwentick, T., Zeume, T.: Reachability is in DynFO. In: Halldórsson, M.M., Iwama, K., Kobayashi, N., Speckmann, B. (eds.) ICALP 2015. LNCS, vol. 9135, pp. 159–170. Springer, Heidelberg (2015). doi:10.1007/978-3-662-47666-6_13

4. Demri, S., Gascon, R.: The effects of bounding syntactic resources on Presburger LTL. J. Logic Comput. **19**(6), 1541–1575 (2009)
5. Dong, G., Jianwen, S.: Incremental and decremental evaluation of transitive closure by first-order queries. Inf. Comput. **120**(1), 101–106 (1995)
6. Gelade, W., Marquardt, M., Schwentick, T.: The dynamic complexity of formal languages. ACM Trans. Comput. Logic (TOCL) **13**(3), 19 (2012)
7. Greenlaw, R., Hoover, H.J., Ruzzo, W.L.: Limits to Parallel Computation: P-Completeness Theory. Oxford University Press, New York (1995)
8. Haase, C., Kreutzer, S., Ouaknine, J., Worrell, J.: Reachability in succinct and parametric one-counter automata. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 369–383. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04081-8_25
9. Hesse, W.: The dynamic complexity of transitive closure is in DynTC^0. Theoret. Comput. Sci. **296**(3), 473–485 (2003)
10. Hesse, W.: Dynamic computational complexity. Ph.D. thesis, Department of Computer Science, University of Massachusetts at Amherst, USA, September 2003
11. Hesse, W., Immerman, N.: Complete problems for dynamic complexity classes. In: Proceedings of the 17th Annual Symposium on Logic in Computer Science (LICS 2002), pp. 313–322. IEEE Comp. Soc. Press, July 2002
12. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison Wesley, Boston (2006)
13. Immerman, N.: Descriptive Complexity. Graduate Texts in Computer Science. Springer, New York (1999)
14. Miltersen, P.B., Subramanian, S., Vitter, J.S., Tamassia, R.: Complexity models for incremental computation. Theoret. Comput. Sci. **130**(1), 203–236 (1994)
15. Muñoz, P., Vortmeier, N., Zeume, T.: Dynamic graph queries. In: Proceedings of the 19th International Conference on Database Theory, ICDT 2016. Leibniz International Proceedings in Informatics, vol. 48, pp. 14:1–14:18. Leibniz-Zentrum für Informatik (2016)
16. Patnaik, S., Immerman, N.: Dyn-FO: a parallel, dynamic complexity class. J. Comput. Syst. Sci. **55**(2), 199–209 (1997)
17. Weber, V., Schwentick, T.: Dynamic complexity theory revisited. Theory Comput. Syst. **40**(4), 355–377 (2007)

Proof Theory

Cyclic Arithmetic Is Equivalent to Peano Arithmetic

Alex Simpson(✉)

Faculty of Mathematics and Physics,
University of Ljubljana, Ljubljana, Slovenia
Alex.Simpson@fmf.uni-lj.si

Abstract. *Cyclic proof* provides a style of proof for logics with inductive (and coinductive) definitions, in which proofs are cyclic graphs representing a form of argument by infinite descent. It is easily shown that cyclic proof subsumes proof by (co)induction. So cyclic proof systems are at least as powerful as the corresponding proof systems with explicit (co)induction rules. Whether or not the converse inclusion holds is a non-trivial question. In this paper, we resolve this question in one interesting case. We show that a cyclic formulation of first-order arithmetic is equivalent in power to Peano Arithmetic. The proof involves formalising the meta-theory of cyclic proof in a subsystem of second-order arithmetic.

1 Introduction

Cyclic (or *circular*[1]) proof has been studied by a number of authors, see, e.g., [1,2,4–13,17–19,21,22,24]. It is a style of proof suitable for logics with inductive and coinductive definitions. The main idea is to allow proofs to be given as cyclic graphs, where the cycles capture the looping nature of arguments by induction and coinduction. For this to provide a sound method of reasoning, a global condition needs to be satisfied by the proof structure in order to rule out fallacious circular arguments. The global condition can be seen as defining cyclic proof as a formalisation of the concept of proof by infinite descent.

In [4,5,10,11], Brotherston and the author studied a natural style of cyclic proof for first-order logic extended with ordinary inductive definitions in the style of Martin-Löf [16]. It was shown that cyclic proof subsumes proof by induction. The question of whether the two styles of proof are equivalent in power was left open, but conjectured to have a positive answer. In a paper appearing alongside this one, Berardi and Tatsuta refute this conjecture [3].[2] In general, cyclic proof is

This material is based upon work supported by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF under Award No. FA9550-14-1-0096.

[1] We prefer *cyclic* for two reasons: proofs are given as cyclic graphs; and the phrase "circular proof" is uncomfortably close to "circular argument", which means an argument that goes round in a circle without establishing anything.

[2] Independently of [3], Stratulat announced a number of potential counterexamples to the conjecture in [23].

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 283–300, 2017.
DOI: 10.1007/978-3-662-54458-7_17

more powerful than proof by induction (as long as the latter uses only induction principles for the inductive definitions under consideration).

In this paper, we provide a complementary result to the theorem of Berardi and Tatsuta. We study cyclic proof for first-order arithmetic. Our main result is that the resulting *Cyclic Arithmetic* coincides with Peano Arithmetic. Thus, in the context of first-order arithmetic, there is, after all, an equivalence in power between cyclic proof and proof by induction.

From one point of view, the study of cyclic proof is particularly apposite in the context of arithmetic. In Sect. 2, we argue that Cyclic Arithmetic is of intrinsic interest as a natural formalisation of the notion of proof by *infinite descent*. Our theorem thus has potential philosophical value in establishing a non-trivial equivalence between infinite descent and induction. More generally, our result contributes to the broad programme of obtaining a better understanding of potential methods of proof. Cyclic methods, in particular, appear to offer a promising extension to machine-assisted formalised proof [9], particularly for applications in computer-science-oriented logics [1,2,6–8,12,13,17–19,24].

In Sect. 3, we introduce an infinitary proof system, whose ∞-*proofs* are non-well-founded trees. This is sound and complete for the first-order theory of true arithmetic. Then in Sect. 4, we define *cyclic proofs* as the restriction of ∞-proofs to regular trees — those that can be presented as finite (cyclic) graphs. Importantly, the question of whether a finite graph presents a cyclic proof is decidable. We end Sect. 4 with the proof that cyclic proof subsumes proof by induction; i.e., that Cyclic Arithmetic contains Peano Arithmetic.

The results thus far are all direct analogues, in the setting of arithmetic, of results in [11] for general inductive definitions. Nevertheless, we give detailed proofs. In the case of the completeness theorem and of the proof that cyclic proof subsumes induction, we do so because the proofs, in the context of arithmetic, are simpler than the corresponding proofs for inductive definitions. In the case of the soundness and decidability results, the proofs for arithmetic are similar to those in [11]. Nevertheless, we supply the details because they are needed in the proof of our main result, Theorem 6, which states that Cyclic Arithmetic is conservative over Peano Arithmetic.

Theorem 6 is proved in Sect. 5. The proof method is to formalise the soundness argument for ∞-proofs in ACA_0, a subsystem of second-order arithmetic, which has been widely studied in the context of reverse mathematics [20]. The use of second-order logic is essential for formalising soundness because of the infinitary nature of ∞-proofs. The reason for the particular choice of the subsystem ACA_0 is that it is conservative over Peano Arithmetic. Once the soundness of Cyclic Arithmetic has been established in ACA_0, the conservativity of Cyclic Arithmetic then follows from a lemma (Lemma 9) that says that ACA_0 can recognise a cyclic proof when presented with one.

Section 6 is devoted to the proof of Lemma 9. For this, we make use of constructions and results from the theory of Büchi automata, once again formalised in ACA_0. Our presentation builds on the recent work of [15], in which

the fundamental complementation result for Büchi automata is studied from the perspective of reverse mathematics.

Finally, in Sect. 7, we discuss directions for further research.

2 Proof by Infinite Descent

The aim of this section is to motivate Cyclic Arithmetic informally as providing a natural style of number-theoretic proof by infinite descent. We begin with a standard example of a proof by infinite descent, establishing that $\sqrt{2}$ is irrational. Since we work in the language of arithmetic, we express this as: there are no natural numbers x_0, x_1 such that $x_0 > 0$ and $x_0^2 = 2x_1^2$.

Suppose, for contradiction, that we have x_0, x_1 such that $x_0 > 0$ and $x_0^2 = 2x_1^2$. It then follows that $x_0 > x_1 > 0$. Since 2 is a prime factor of x_0^2 it must be a prime factor of x_0 itself. So $x_0 = 2x_2$ for some x_2. So $4x_2^2 = 2x_1^2$, hence $x_1^2 = 2x_2^2$.

We have now gone round in a circle back to the start of the proof, but with x_1, x_2 in place of x_0, x_1. By repeating the argument for x_1, x_2 we discover that $x_1 > x_2 > 0$ and $x_2^2 = 2x_3^2$ for some x_3. Continuing, $x_2 > x_3 > 0$ and $x_3^2 = 2x_4^2$; whence $x_3 > x_4 > 0$, etc. So, starting from our initial assumptions that $x_0 > 0$ and $x_0^2 = 2x_1^2$, we produce an infinite strictly descending sequence $x_0 > x_1 > x_2 > x_3 > \ldots$ of positive integers. Since no such sequence exists, we have obtained the desired contradiction.

Figure 1 presents this proof as an infinite proof tree of sequents. The steps labelled ($\star$) and (†) are not intended to be atomic proof steps. Rather ($\star$) is the main number-theoretic lemma used in the proof, and (†) chains together a few simple steps of arithmetical and logical reasoning (including a cut). The ellipsis at the top right represents the continuation of the argument via an infinite sequence of repetitions of the visible proof pattern, but with the variables changed appropriately at each repetition.

$$\dfrac{\dfrac{}{0<x_0,\ x_0^2 = 2x_1^2 \Rightarrow 0<x_1<x_0 \wedge \exists x_2.\, x_0 = 2x_2}\,(\star) \qquad \dfrac{\vdots}{\dfrac{0<x_1,\ x_1^2 = 2x_2^2 \Rightarrow \bot}{\underline{x_1<x_0},\ 0<x_1,\ 4x_2^2 = 2x_1^2 \Rightarrow \bot}}}{0<x_0,\ x_0^2 = 2x_1^2 \Rightarrow \bot}\,(\dagger)$$

Fig. 1. Infinite descent proof of the irrationality of $\sqrt{2}$

The proof tree in its entirety is an infinite tree, with one growing up to the right. Going up this branch, the variables $x_0, x_1, x_2, \ldots$, once introduced, never change their value. Moreover, we pass through an infinite sequence of underlined statements $x_1 < x_0$, $x_2 < x_1$, $x_3 < x_2$ each appearing as an antecedent (i.e., left-hand formula) in a sequent. It is this fact that makes the argument a valid proof by infinite descent.

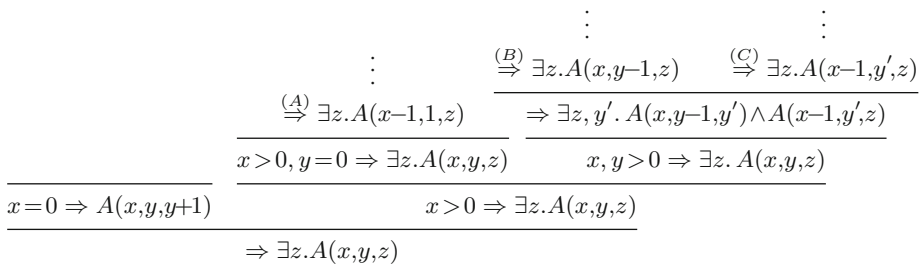


Fig. 2. Infinite descent proof of the totality of the Ackermann-Péter function.

We present one more example of a proof in a similar style. Let $A(x, y, z)$ be a ternary relation on natural numbers satisfying:

$$\begin{aligned} A(x,y,z) \Leftrightarrow\ & (x = 0 \wedge z = y + 1) \\ & \vee\ (x > 0 \wedge y = 0 \wedge A(x-1, 1, z)) \\ & \vee\ (x, y > 0 \wedge \exists w.\, A(x, y-1, w) \wedge A(x-1, w, z)) \end{aligned}$$

This formula defines A to be the graph of the well-known 2-argument Ackermann-Péter function. Using standard techniques of definition, one can encode $A(x, y, z)$ by a Σ^0_1-formula in the language of arithmetic satisfying the equivalence above.

Figure 2 presents a proof by infinite descent of the totality of the Ackermann-Péter function. As before, the individual rules are not atomic steps, but contain arithmetical and logical reasoning, including manipulation of the defining property of $A(x, y, z)$ above. This time, the full infinite proof is built by repeating the basic pattern three times, once each at (A), (B) and (C), *ad infinitum*. In doing this, variables are substituted by the terms specified in each case. Note that infinitely many variables $y, y', y'', \ldots$ appear in the infinite proof.

The proof in Fig. 2 presents a correct argument by infinite descent for the following reason. By the local soundness of all rules in the proof, any x, y providing a counterexample to the concluding sequent will generate an infinite branch together with assignments to all variables such that all sequents along the branch are false. There are now two cases to consider.

- If the infinite branch passes through sequents in positions (A) or (C) infinitely often then the value of the number supplied in the x position of $A(x, y, z)$ is decremented infinitely often even though it remains positive along the path.
- Otherwise, the branch must eventually reach a point after which it avoids (A) and (C). Thus all subsequent repetitions are attained via (B). In this case, the number appearing in the y position, at the point in question, is decremented infinitely often, although it again remains positive.

In either case, we have the desired contradiction.

Thus far, we have been describing proofs by infinite descent as infinite proofs. This is in keeping with the idea that a proof by infinite descent should construct infinite sequences, but it is not compatible with the idea of a proof as a finite representation of an argument. Nonetheless, a very natural restriction on infinite proofs can be imposed to achieve such a finite representation. One simply asks for the infinite proof tree to be *regular*, namely for it to have only finitely many distinct subtrees. Any such regular infinite proof tree can be presented as a finite cyclic graph. For example, consider the version below of the proof from Fig. 1, in which a substitution rule is used to make all subtrees rooted at sequents labelled $(*)$ identical. The full proof tree is thus presented by the finite cyclic graph obtained by identifying the nodes labelled $(*)$.

$$\dfrac{\dfrac{}{0<x_0,\, x_0^2 = 2x_1^2 \Rightarrow 0<x_1<x_0 \wedge \exists x_2.\, x_0 = 2x_2} \qquad \dfrac{\dfrac{\vdots}{0<x_0,\, x_0^2 = 2x_1^2 \overset{(*)}{\Rightarrow} \bot}}{0<x_1,\, x_1^2 = 2x_2^2 \Rightarrow \bot}\,(\mathsf{Sub})}{\underline{x_1<x_0},\, 0<x_1,\, 4x_2^2 = 2x_1^2 \Rightarrow \bot}$$

$$0<x_0,\, x_0^2 = 2x_1^2 \overset{(*)}{\Rightarrow} \bot$$

It is similarly possible to convert Fig. 2 to a regular infinite proof.

In Sect. 3, we give a precise definition of ∞-*proof* that formalises the notion of infinite proof by infinite descent described informally above, and we show that ∞-proofs are sound and complete for the first-order theory of true arithmetic. Since the notion of proof is infinitary, the completeness result is unsurprising. For example, a similar completeness property is well known to hold for ω-*proofs*, obtained by adding the infinitary ω-rule to (for example) sequent calculus. Nonetheless, there is an important mathematical distinction between ω-proofs and ∞-proofs. The former are given as infinitely-branching well-founded trees. In contrast, ∞-proofs are finitely branching (potentially) non-well-founded trees.

It is an advantage of ∞-proofs that they possess a naturally identifiable subclass of finitely presentable proofs, the *regular* ones, which we introduce as *cyclic proofs* in Sect. 4. Our main result, the coincidence of cyclic proof for arithmetic with Peano Arithmetic, thus establishes that *finitary proof by infinite descent is equivalent to proof by induction*.

3 ∞-proofs

We formulate arithmetic using first-order logic with equality, with signature $(0, s, +, \cdot, <)$, where s is the successor function. The strict order relation $<$ is included as primitive because it is used in the definition of ∞-proof below.

We give a sequent calculus presentation of our proof calculus. For our purposes, a *sequent* $\Gamma \Rightarrow \Delta$ is a pair of finite sets Γ, Δ of formulas. We use standard

notational conventions for sequents, such as omitting set delimiters when writing sets, and using comma ',' for union. We write $\Gamma[\theta]$ for the result of applying the same substitution θ (mapping finitely many variables to associated terms) to every formula in Γ. We also write $\Gamma[t_1, \ldots, t_k]$, for terms $t_1, \ldots, t_k$, to mean $\Gamma[t_1/x_1, \ldots, t_k/x_k]$, where $x_1, \ldots, x_k$ are distinct variables left implicit. In such cases, a parallel mention of $\Gamma[u_1, \ldots, u_k]$ always means $\Gamma[u_1/x_1, \ldots, u_k/x_k]$ for the same variables $x_1, \ldots, x_k$.

Our proof system is built from three sets of rules manipulating sequents. Rules for the logical constants (including equality) are presented in Fig. 3. Structural rules, including (Cut), are given in Fig. 4. Finally, basic arithmetic properties are axiomatised, in Fig. 5, by a list of 11 axiom sequents, together with one further inference rule. The axioms and rule of Fig. 5 implement a finitely axiomatised theory of arithmetic corresponding to a natural expansion of Robinson's system Q with $<$ as a primitive relation. As motivated in Sect. 2, our ∞-proofs are infinite trees, where locally each node of the tree is given by an application of one of the rules or axioms in Figs. 3, 4 and 5. We call such a tree a *pre-∞-proof*. In order to qualify as an actual proof, such a tree will need to satisfy a further condition, whose formulation requires the following definition.

$$\frac{}{\Gamma \Rightarrow \Delta}\ \Gamma \cap \Delta \neq \emptyset \qquad \frac{\Gamma \Rightarrow A, \Delta}{\Gamma, \neg A \Rightarrow \Delta} \qquad \frac{\Gamma, A \Rightarrow \Delta}{\Gamma \Rightarrow \neg A, \Delta}$$

$$\frac{\Gamma, A, B \Rightarrow \Delta}{\Gamma, A \wedge B \Rightarrow \Delta} \qquad \frac{\Gamma \Rightarrow A, \Delta \quad \Gamma \Rightarrow B, \Delta}{\Gamma \Rightarrow A \wedge B, \Delta} \qquad \frac{\Gamma, A \Rightarrow \Delta \quad \Gamma, B \Rightarrow \Delta}{\Gamma, A \vee B \Rightarrow \Delta} \qquad \frac{\Gamma \Rightarrow A, B, \Delta}{\Gamma \Rightarrow A \vee B, \Delta}$$

$$\frac{\Gamma, A[t/x] \Rightarrow \Delta}{\Gamma, \forall x\, A \Rightarrow \Delta} \qquad \frac{\Gamma \Rightarrow A[y/x], \Delta}{\Gamma \Rightarrow \forall x\, A, \Delta}\ y \text{ fresh} \qquad \frac{\Gamma, A[y/x] \Rightarrow \Delta}{\Gamma, \exists x\, A \Rightarrow \Delta}\ y \text{ fresh} \qquad \frac{\Gamma \Rightarrow A[t/x], \Delta}{\Gamma \Rightarrow \exists x\, A, \Delta}$$

$$\frac{\Gamma[u_1, u_2] \Rightarrow \Delta[u_1, u_2]}{\Gamma[u_2, u_1],\ u_1 = u_2 \Rightarrow \Delta[u_2, u_1]} \qquad \frac{}{\Gamma \Rightarrow t = t,\ \Delta}$$

Fig. 3. Cut-free sequent calculus with equality

$$\frac{\Gamma \Rightarrow \Delta}{\Gamma, \Gamma' \Rightarrow \Delta', \Delta}\ (\mathsf{Wk}) \qquad \frac{\Gamma,\ A \Rightarrow \Delta \quad \Gamma \Rightarrow A,\ \Delta}{\Gamma \Rightarrow \Delta}\ (\mathsf{Cut}) \qquad \frac{\Gamma \Rightarrow \Delta}{\Gamma[\theta] \Rightarrow \Delta[\theta]}\ (\mathsf{Sub})$$

Fig. 4. Weakening, cut and substitution rules

Definition 1 (Precursor, trace, progress). Let $(\Gamma_i \Rightarrow \Delta_i)_{i \geq 0}$ be an infinite branch through a pre-proof. For terms t, t', we say that t' is a *precursor* of t at i if one of the following holds.

$$
\begin{aligned}
t < u,\ u < v &\Rightarrow t < v \\
t < u,\ u < t &\Rightarrow \\
&\Rightarrow t < u,\ t = u,\ u < t \\
t < 0 &\Rightarrow \\
t < u &\Rightarrow \mathsf{s}(t) < \mathsf{s}(u) \\
&\Rightarrow t < \mathsf{s}(t) \\
t < u,\ u < \mathsf{s}(t) &\Rightarrow \\
&\Rightarrow t + 0 = t \\
&\Rightarrow t + \mathsf{s}(u) = \mathsf{s}(t + u) \\
&\Rightarrow t \cdot 0 = 0 \\
&\Rightarrow t \cdot \mathsf{s}(u) = (t \cdot u) + t
\end{aligned}
$$

$$
\frac{\Gamma, t = \mathsf{s}(x) \Rightarrow \Delta}{\Gamma, 0 < t \Rightarrow \Delta}\ x \text{ fresh}
$$

Fig. 5. Axioms and rules for basic arithmetic

- $\Gamma_i \Rightarrow \Delta_i$ is the conclusion of an application of (Sub), and $t = \theta(t')$ where θ is the substitution used in the rule application.
- $\Gamma_i \Rightarrow \Delta_i$ is the conclusion of an $(u_1 = u_2)$-left rule, and it is possible to write t' and t as $u[u_1, u_2]$ and $u[u_2, u_1]$ respectively, for some term u.
- $\Gamma_i \Rightarrow \Delta_i$ is the conclusion of one of the other rules, and $t' = t$.

We say that a term t *occurs* in a sequent $\Gamma \Rightarrow \Delta$ if it appears within some formula in Γ, Δ (possibly as a subterm of another term). A *trace* along $(\Gamma_i \Rightarrow \Delta_i)_{i \geq 0}$ is a sequence $(t_i)_{i \geq N}$, for some $N \geq 0$, such that, for every $i \geq N$, the term t_i occurs in $\Gamma_i \Rightarrow \Delta_i$, and also one of the following holds.

- Either t_{i+1} is a precursor of t_i at i,
- or there exists $(t_{i+1} < t) \in \Gamma_{i+1}$ such that t is a precursor of t_i at i.

When the latter case holds, we say that the trace *progresses* at $i + 1$.

Definition 2 (∞-proof). An ∞-*proof* is a pre-∞-proof that satisfies the following *trace condition*.

> Along every infinite branch $(\Gamma_i \Rightarrow \Delta_i)_i$ there exist $N \geq 0$ and a trace $(t_i)_{i \geq N}$ that progresses at infinitely many i.

Modulo the expansion of the depicted rules into combinations of primitive rules from Figs. 3, 4 and 5, the proofs in Figs. 1 and 2 are both ∞-proofs, for the reasons explained in Sect. 2. (E.g., the required trace in Fig. 1 is $x_0, x_1, x_1, x_2, x_2, \ldots$.)

Semantically, we will be interested only in the standard interpretation in the natural numbers $\mathbb{N}$. We write $\mathbb{N} \models^{\rho} A$ to say that formula A is true in $\mathbb{N}$ under

an environment ρ that interprets the free variables of A as natural numbers. We write $\mathbb{N} \models^\rho \Gamma \Rightarrow \Delta$ to mean: if $\mathbb{N} \models^\rho A$ for all $A \in \Gamma$ then there exists $B \in \Delta$ such that $\mathbb{N} \models^\rho B$. We define $\mathbb{N} \models \Gamma \Rightarrow \Delta$ to mean: $\mathbb{N} \models^\rho \Gamma \Rightarrow \Delta$ for all $\mathbb{N}$-environments ρ.

Theorem 3 (Soundness for ∞-proofs). *If $\Gamma \Rightarrow \Delta$ has an ∞-proof then $\mathbb{N} \models \Gamma \Rightarrow \Delta$.*

Proof. We suppose, for contradiction, that we have an ∞-proof of $\Gamma_0 \Rightarrow \Delta_0$, but that $\mathbb{N} \not\models^{\rho_0} \Gamma_0 \Rightarrow \Delta_0$.

We first construct an infinite branch $(\Gamma_i \Rightarrow \Delta_i)_i$, together with an associated sequence $(\rho_i)_i$ of environments, such that $\mathbb{N} \not\models^{\rho_i} \Gamma_i \Rightarrow \Delta_i$ for all i. To do this, $\Gamma_{i+1} \Rightarrow \Delta_{i+1}$ and ρ_{i+1} are constructed from $\Gamma_i \Rightarrow \Delta_i$ and ρ_i as follows. Since $\mathbb{N} \not\models^{\rho_i} \Gamma_i \Rightarrow \Delta_i$, the sequent $\Gamma_i \Rightarrow \Delta_i$ must be the conclusion of an inference rule. If this rule is an instance of (Sub) then define $\rho_{i+1} = \rho_i \circ \theta$, where θ is the substitution in the rule. Otherwise define $\rho_{i+1} = \rho_i$. By the soundness of inference rules, at least one premise of the rule is a sequent $\Gamma' \Rightarrow \Delta'$ for which $\mathbb{N} \not\models^{\rho_{i+1}} \Gamma' \Rightarrow \Delta'$. We define $\Gamma_{i+1} \Rightarrow \Delta_{i+1}$ to be a chosen such premise.

By the trace condition, the infinite branch has a trace $(t_i)_{i \geq N}$ that progresses infinitely often. Consider the associated sequence of numbers $(t_i^{\rho_i})_{i \geq N}$. Since $\mathbb{N} \not\models^{\rho_i} \Gamma_i \Rightarrow \Delta_i$, we have that $\mathbb{N} \models^{\rho_i} A$ for every $A \in \Gamma_i$. So, using the definitions of precursor and of ρ_{i+1}, if t_{i+1} is a precursor of t then ${t_{i+1}}^{\rho_{i+1}} = t^{\rho_i}$. Therefore,

- $t_{i+1}^{\rho_{i+1}} = t_i^{\rho_i}$, if t_{i+1} is a precursor of t_i; and
- $t_{i+1}^{\rho_{i+1}} < t_i^{\rho_i}$, if $(t_{i+1} < t) \in \Gamma_{i+1}$ and t is a precursor of t_i.

By the trace condition, the second case applies infinitely often. Thus $(t_i^{\rho_i})_{i \geq N}$ is an infinite non-increasing sequence of natural numbers that decreases infinitely often, which gives the desired contradiction. □

Due to their infinitary nature, ∞-proofs are complete. Indeed, completeness holds even for proofs that contain no instances of (Wk) and (Sub), and in which (Cut) occurs only in cases in which the cut formula A is atomic. We call such proofs *atomic cut* ∞-proofs.

Theorem 4 (Atomic-cut completeness for ∞-proofs). *If $\mathbb{N} \models \Gamma \Rightarrow \Delta$ then there exists an atomic-cut ∞-proof of $\Gamma \Rightarrow \Delta$.*

Proof. The main observation required is that the sequent calculus ω-rule:

$$\frac{\Gamma[0] \Rightarrow \Delta[0] \quad \Gamma[s(0)] \Rightarrow \Delta[s(0)] \quad \Gamma[s(s(0))] \Rightarrow \Delta[s(s(0))] \quad \ldots \quad \ldots \quad \ldots}{\Gamma[t] \Rightarrow \Delta[t]}$$

is simulated by the ∞-proof below (we combine multiple rules into single steps).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma[0] \Rightarrow \Delta[0]}}{t = 0, \Gamma[t] \Rightarrow \Delta[t]} \qquad \dfrac{\dfrac{}{t = s(x_1), \Gamma[t] \Rightarrow \Delta[t]}}{t > 0, \Gamma[t] \Rightarrow \Delta[t]}}{\Gamma[t] \Rightarrow \Delta[t]} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma[s(0)] \Rightarrow \Delta[s(0)]}}{x_1 = 0, \Gamma[s(x_1)] \Rightarrow \Delta[s(x_1)]} \qquad \dfrac{\dfrac{}{x_1 = s(x_2), \Gamma[s(x_1)] \Rightarrow \Delta[s(x_1)]}}{x_1 > 0, \Gamma[s(x_1)] \Rightarrow \Delta[s(x_1)]}}{\underline{x_1 < t}, \Gamma[s(x_1)] \Rightarrow \Delta[s(x_1)]} \qquad \dfrac{\dfrac{\dfrac{}{\Gamma[s(s(0))] \Rightarrow \Delta[s(s(0))]} \qquad \dfrac{\vdots}{x_2 > 0, \Gamma[s(s(x_2))] \Rightarrow \Delta[s(s(x_2))]}}{\underline{x_2 < x_1}, \Gamma[s(s(x_2))] \Rightarrow \Delta[s(s(x_2))]}}{\underline{x_1 < t}, \Gamma[s(x_1)] \Rightarrow \Delta[s(x_1)]}}{\Gamma[t] \Rightarrow \Delta[t]}$$

To apply the above, we plug in an ∞-proof for each premise $\Gamma[\underline{n}] \Rightarrow \Delta[\underline{n}]$. The resulting pre-$\infty$-proof has just one additional infinite branch, the rightmost branch. Along this branch, the sequence $t, t, t, x_1, x_1, x_1, x_2, x_2, x_2, \ldots$ is an infinitely progressing trace. (The progress points are underlined.) □

4 Cyclic Arithmetic

A (possibly infinite) tree is said to be *regular* if it has only finitely many distinct subtrees. Equivalently, a tree is regular if it can be defined as an unfolding of a finite directed (possibly cyclic) graph.

We shall be interested in regular ∞-proofs; that is, in ∞-proofs whose underlying pre-∞-proofs are regular trees. For this, we consider a regular pre-∞-proof to be presented by a finite graph of the following form. Each vertex v of the graph is labelled with an instance $\mathsf{rule}(v)$ of one of the rules in Figs. 3, 4 and 5. We write $\mathsf{conc}(v)$ for the sequent that is the conclusion of the rule instance, and $\mathsf{prem}_i(v)$ for the sequent that is the i-th premise. (Axioms are considered as rules with 0 premises.) When a vertex v has label $\mathsf{rule}(v)$ with n premises, the graph must contain exactly n edges $e_v^1, \ldots, e_v^n$ with source v. Moreover, for each $i = 1, \ldots, n$, it must hold that $\mathsf{prem}_i(v) = \mathsf{conc}(v_i)$, where v_i is the target vertex of e_v^i. Finally, there is a distinguished vertex ε, which represents the conclusion of the pre-∞-proof; i.e., $\mathsf{conc}(\varepsilon)$ is the conclusion sequent.

It is straightforward to see how each such finite graph presentation unfolds to a regular pre-∞-proof, and conversely how each regular pre-∞-proof can be given such a presentation; see [11] for a detailed treatment.

We next establish the decidability of the global trace condition for regular pre-∞-proofs, for which we follow analogous arguments in [11,21]. Although the relatively simple construction does not provide a practical decision procedure, it has the advantage of facilitating the proofs in Sect. 6 below.

Recall that a *(nondeterministic) Büchi automaton* over an alphabet Σ is just a nondeterministic finite automaton over Σ with an acceptance condition defined for *infinite* words as follows. An *accepting run* of a Büchi automaton B is an infinite sequence of consecutive transitions, starting from an initial state,

that passes through an accepting state infinitely often. An infinite word $X \in \Sigma^\omega$ is *accepted* by B if there exists an accepting run labelled by X.

Theorem 5. *It is decidable whether a regular pre-∞-proof, presented as a finite directed graph, is an ∞-proof.*

Proof. Let Σ be the alphabet whose symbols are the edges in the finite graph G that presents the proof. We construct a Büchi automaton B_t over Σ that recognises those infinite paths through G that possess an infinitely progressing trace. The states of B_t are of the form:

- (v) where v is a vertex in G.
- (v, t) where v is a vertex in G and t is a term in $\mathsf{conc}(v)$.
- $(v, t, \checkmark)$ where v is a vertex in G and t is a term in $\mathsf{conc}(v)$.

The transitions are of the form:

- $(v) \xrightarrow{e^i_v} (v')$ and $(v) \xrightarrow{e^i_v} (v', t')$ whenever v' is the target of e^i_v.
- $(v, t) \xrightarrow{e^i_v} (v', t')$ and $(v, t, \checkmark) \xrightarrow{e^i_v} (v', t')$ whenever v' is the target of e^i_v and t' is a precursor of t for $\mathsf{rule}(v)$.
- $(v, t) \xrightarrow{e^i_v} (v', t', \checkmark)$ and $(v, t, \checkmark) \xrightarrow{e^i_v} (v', t', \checkmark)$ whenever v' is the target of e^i_v and $(t' < t'')$ is an antecedent of $\mathsf{conc}(v')$ for some precursor t'' of t for $\mathsf{rule}(v)$.

The accepting states are those with a $\checkmark$ component. The start state is ε. It is clear that an ω-word is accepted if and only if it defines an infinite path through the pre-∞-proof that possess an infinitely progressing trace.

We also need a Büchi automaton B_p that recognises the language of all infinite paths through the pre-∞-proof. The construction of this is trivial, hence omitted.

The pre-∞-proof is an ∞-proof if and only if there is an inclusion of ω-languages $\mathcal{L}(B_p) \subseteq \mathcal{L}(B_t)$. Such inclusions are decidable. □

Since regular ∞-proofs are presented as finite cyclic graphs, we call such proofs *cyclic proofs*, following the terminology of [11]. Similarly, we call the first-order theory consisting of all sentences A such that the sequent $\Rightarrow A$ has a cyclic proof *Cyclic Arithmetic* (CA).

The main result of this paper is that Cyclic Arithmetic coincides with Peano Arithmetic (PA). To ease the comparison, we assume that PA is also formulated in the language $(0, s, +, \cdot, <)$.

Theorem 6 (Coincidence theorem). $\mathsf{CA} = \mathsf{PA}$.

The proposition below establishes the easier inclusion of Theorem 6. The converse inclusion is left to Sect. 5.

Proposition 7. $\mathsf{PA} \subseteq \mathsf{CA}$.

Proof. The axioms and rule of Figs. 3, 4 and 5 capture all of PA except for the induction schema. Moreover, since we have the cut rule, the cyclic-provable sentences are closed under logical consequence. We thus need only show that every instance of induction has a cyclic proof. Such a proof is given by (we identify the $(*)$ nodes):

$$\dfrac{\dfrac{}{x_0 = 0, A[0] \Rightarrow A[x_0]} \qquad \dfrac{\dfrac{\dfrac{\dfrac{A[0], \forall y.(A[y] \to A[s(y)]) \overset{(*)}{\Rightarrow} A[x_0]}{A[0], \forall y.(A[y] \to A[s(y)]) \Rightarrow A[x_1]}\,(\mathsf{Sub})}{\underline{x_1 < x_0}, A[0], \forall y.(A[y] \to A[s(y)]) \Rightarrow A[s(x_1)]}}{x_0 = s(x_1), A[0], \forall y.(A[y] \to A[s(y)]) \Rightarrow A[x_0]}}{x_0 > 0, A[0], \forall y.(A[y] \to A[s(y)]) \Rightarrow A[x_0]}}{A[0], \forall y.(A[y] \to A[s(y)]) \overset{(*)}{\Rightarrow} A[x_0]}$$

The infinite trace is $(x_0\, x_0\, x_0\, x_1\, x_1\, x_1)^\omega$. This indeed progresses infinitely often, at the point underlined in the proof. □

5 Conservativity of CA over PA

The goal of this section is to prove that $\mathsf{CA} \subseteq \mathsf{PA}$. The first step is to prove the soundness of ∞-proofs in ACA_0, a well-known subsystem of second-order arithmetic, which enjoys the property of being conservative over PA. The use of a second-order language allows the formalisation of concepts associated with infinite trees and infinite paths through them, which is necessary for reasoning about ∞-proofs.

Recall (see [20] for a detailed exposition) that the language of second-order arithmetic extends our first-order language with: set variables $X, Y, Z, \ldots$; with quantification over set variables; and with a new atomic formula $t \in X$, where t is a first-order term and X a set variable. A formula is said to be *arithmetical* if it does not contain any set quantifiers (it may contain free set variables). The theory ACA_0 contains the usual first-order axioms of arithmetic, the expected quantifier rules for set variables, and the two principles below.

– The *induction axiom*:

$$\forall X.\ 0 \in X \wedge (\forall x.\, x \in X \to s(x) \in X) \to \forall x.\, x \in X.$$

– The *arithmetical comprehension schema*:

$$\exists X. \forall x. (x \in X \leftrightarrow \phi),$$

where ϕ ranges over arithmetical formulas in which the set variable X does not occur free.

We assume reasonable encodings of ordered pairs, and sequences as numbers, in which each ordered pair and sequence has a unique encoding. A set X of natural numbers *encodes a tree* if:

- every $x \in X$ encodes a sequence $x_1 \dots x_k$ for some $k \geq 0$;
- if (an encoding of) $x_1 \dots x_k x_{k+1} \in X$ then also $x_1 \dots x_k \in X$; and
- $\varepsilon \in X$ (where ε encodes the empty sequence).

A (partial) function is encoded as a set X of (codes of) ordered pairs satisfying: if $(x, y) \in X$ and $(x, z) \in X$ then $y = z$. The *domain* of X is $\{x \mid \exists y.\ (x, y) \in X\}$. We say that a set X *encodes a labelled tree* if it encodes a function whose domain is a tree. In this case we call the elements of the domain of X the *nodes* of X, and we call the result of applying the function to a node x the *label* of x.

We shall encode ∞-proofs as trees labelled with Gödel numbers of instances of rules from Figs. 3, 4 and 5. For this, we assume a reasonable Gödel numbering of terms, formulas, sequents and rule instances.

Let X encode a labelled tree. We say that X *encodes a pre-∞-proof* if:

- for every $(x, y) \in X$, we have that y encodes a valid rule instance $\mathsf{rule}(x)$;
- if $\mathsf{rule}(x_1 \dots x_k)$ has n premises then the n sequences $x_1 \dots x_k 1, \dots, x_1 \dots x_k n$ are all nodes in X, and no other sequence $x_1 \dots x_k i$ is a node in X; and
- if $x_1 \dots x_k i$ is a node in X then $\mathsf{prem}_i(x_1 \dots x_k) = \mathsf{conc}(x_1 \dots x_k i)$, where we write $\mathsf{prem}_i(x)$ for the i-th premise of $\mathsf{rule}(x)$ and $\mathsf{conc}(x)$ for the conclusion.

An *infinite branch* through a tree X is given by a function Y with domain $\mathbb{N}$, such that $Y(0) = \varepsilon$ and, for every x, it holds that $Y(s(x))$ is a child in X of $Y(x)$. A set Z encodes a sequence $(t_i)_{i \geq N}$ of terms if it is a partial function with domain $\{x \mid x \geq N\}$ and, for every $x \geq N$, it holds that $Z(x)$ is the Gödel number of a term. If X encodes a pre-∞-proof, Y is an infinite branch through X, and Z is a sequence of terms then Definition 1 can be directly translated into the language of second-order arithmetic to define (arithmetical) formulas that express each of the properties:

- Z is a trace along Y in X;
- the trace Z progresses at i in the branch Y of X;
- the trace Z progresses infinitely often in the branch Y of X.

(Note that X needs to appear explicitly in the above formulas because the labelling containing the information about which inference rules are applied is present only in X.) One can thus directly formalize Definiton 2 to obtain a (non-arithmetical) formula expressing:

- X is an ∞-proof.

We wish to prove the soundness of ∞-proofs in ACA_0. Thus, we would like to show that, when we have an ∞-proof of a sequent, that sequent is true (under any interpretation of its free variables). However, the above statement cannot be

formulated in ACA_0, because truth is a non-arithmetical property of first-order-arithmetic formulas. We circumvent this by bounding the logical complexity of formulas. For every $n \geq 0$, there is a first-order-arithmetic formula $\mathsf{Tr}_n(x, y)$ which holds if and only if: x is the Gödel number $\ulcorner A \urcorner$ of a Σ^0_n-formula A, and y is the encoding $\ulcorner \rho \urcorner$ an environment ρ, assigning numbers to all free variables in A, such that $\mathbb{N} \models^\rho A$. (See [14, Chap. 9] for a careful construction of such a formula.) If A is a Σ^0_n-sentence then PA proves the truth-reflection schema:

$$\mathsf{Tr}_n(\ulcorner A \urcorner, \ulcorner \emptyset \urcorner) \;\leftrightarrow\; A. \tag{1}$$

Lemma 8 (Formalised soundness). *For every $n \geq 0$, ACA_0 proves: "*for all X, if X is an ∞-proof, containing formulas of complexity at most Σ^0_n, then, for every assignment ρ of numbers to all free variables in the conclusion sequent $\Gamma \Rightarrow \Delta$ of X, the formula $\bigwedge \Gamma \to \bigvee \Delta$ is Σ^0_{n+1}-true under ρ*".*

In the statement of the lemma, we use quoted sans-serif to emphasise the scope of the formal statement proved in ACA_0. Notice that this formal statement is really a statement about encodings of ∞-proofs and Gödel numbers of sequents and formulas. We have stated it informally for readability. Note also that the use of the Σ^0_{n+1}-truth predicate is appropriate because $\bigwedge \Gamma \to \bigvee \Delta$ is a Δ^0_{n+1}-formula. For convenience, we henceforth write the sequent $\Gamma \Rightarrow \Delta$ in place of the formula $\bigwedge \Gamma \to \bigvee \Delta$ in order to talk directly about truth and falsity of sequents.

Proof. We formalise the proof of Theorem 3 in ACA_0. So again suppose we have an ∞-proof X of $\Gamma_0 \Rightarrow \Delta_0$, but that $\Gamma_0 \Rightarrow \Delta_0$ is Σ^0_{n+1}-false under ρ_0.

The main point that requires elaboration is the construction, in ACA_0, of the infinite path $(\Gamma_i \Rightarrow \Delta_i)_i$ in combination with the associated sequence $(\rho_i)_i$ of environments. For this, we assign, to every node x in X, an environment ρ_x:

- $\rho_\varepsilon = \rho_0$.
- $\rho_{x_1 \dots x_k x_{k+1}} = \begin{cases} \rho_{x_1 \dots x_k} & \text{if } \mathsf{rule}(x_1 \dots x_k) \text{ is not an instance of } (\mathsf{Sub}) \\ \rho_{x_1 \dots x_k} \circ \theta & \text{if } \mathsf{rule}(x_1 \dots x_k) \text{ is a } (\mathsf{Sub}) \text{ with substitution } \theta \end{cases}$

Let Y be the set (whose definition, for unbounded n, requires full arithmetical comprehension):

$$\{x \in X \mid \text{ for all prefixes } x' \text{ of } x, \text{ the sequent } \mathsf{conc}(x') \text{ is } \Sigma^0_{n+1}\text{-false under } \rho_{x'}\}.$$

Then Y is a finitely branching tree. Moreover, by the soundness of inference rules, every $x \in Y$ has a child node $x' \in Y$. Thus Y is infinite. Hence, by König's lemma (which is a theorem of ACA_0 [20]), there exists an infinite branch Z in Y. Then $i \mapsto \mathsf{conc}(Z(i))$ defines $(\Gamma_i \Rightarrow \Delta_i)_i$ and $i \mapsto \rho_{Z(i)}$ defines $(\rho_i)_i$.

The remainder of the proof, which argues that an infinitely progressing trace along Z contradicts the falsity of the sequents in Z, goes through exactly as in the proof of Theorem 3. □

We next consider cyclic proofs. Any such is presented by a finite graph of the kind introduced at the start of Sect. 4. We assume a sensible encoding of such

graphs as natural numbers, and we write $\ulcorner G \urcorner$ for the number representing G. Given $\ulcorner G \urcorner$, we define in ACA_0 the pre-∞-proof generated by G as the set $\mathsf{unfold}(\ulcorner G \urcorner)$ defined as:

$$\{(x_1 \dots x_k,\, y) \mid \text{there is a path } \varepsilon\, e^{x_1}\, v_1\, e^{x_2}\, v_2 \dots e^{x_k}\, v_k \text{ in } G, \text{ and } y = \mathsf{rule}(v_k)\}.$$

This indeed defines a set since the defining formula is arithmetical.

Lemma 9. *If G is a finite graph presentation of a regular ∞-proof then* ACA_0 *proves "*$\mathsf{unfold}(G)$ is an ∞-proof*".*

Here, and henceforth, we write $\mathsf{unfold}(G)$ rather than $\mathsf{unfold}(\ulcorner G \urcorner)$, since the latter is the default interpretation of the former. Whenever we refer to a finite combinatorial object within ACA_0, we always do so via its numerical encoding.

We postpone the proof of Lemma 9 to Sect. 6 below. Modulo this one pending proof, we now have all the ingredients to complete the proof of Theorem 6.

Proposition 10. $\mathsf{CA} \subseteq \mathsf{PA}$.

Proof. Suppose G is a finite graph presentation of a regular ∞-proof with conclusion sequent $\Rightarrow A$, where A is a sentence. Let n be such that every formula in G is at most Σ^0_n. By Lemma 9, ACA_0 proves "$\mathsf{unfold}(G)$ is an ∞-proof". Whence, by Lemma 8, ACA_0 proves "A is Σ^0_{n+1}-true". Therefore, by an application of the reflection property (1) of the Σ^0_{n+1}-truth predicate, ACA_0 proves A. Since ACA_0 is conservative over PA, it follows that PA proves A. □

Propositions 7 and 10 together establish Theorem 6.

6 Büchi Automata in ACA_0

It remains to prove Lemma 9. Our method of proof is to apply the theory of Büchi automata as formalised in ACA_0. Since a Büchi automaton B over a finite alphabet Σ is a finite combinatorial object, it can be encoded as a natural number $\ulcorner B \urcorner$. We thus formalise properties of Büchi automata as properties of their codes, but we continue with our policy of omitting explicit reference to codes. In ACA_0, an infinite word is coded as a set X defining an infinite sequence of elements of Σ. The property:

$$B \text{ accepts the infinite word } X$$

is directly expressible by a (non-arithmetical) formula in second-order arithmetic. Using this, we formalise the following properties of Büchi automata in ACA_0.

Proposition 11 (Formalised Büchi intersection). *There is a computable binary function $\sqcap$ such that* ACA_0 *proves: "*for all Büchi automata B_1, B_2, it holds that $B_1 \sqcap B_2$ is a Büchi automaton satisfying:

$$\text{for every infinite } \Sigma\text{-word } X,\ B_1 \sqcap B_2 \text{ accepts } X \leftrightarrow B_1 \text{ and } B_2 \text{ both accept } X''.$$

Proposition 12 (Formalised Büchi complementation). *There is a computable unary function* $(\cdot)^c$ *such that* ACA_0 *proves: "*for every Büchi automaton B it holds that B^c is a Büchi automaton satisfying:

$$\text{for every infinite } \Sigma\text{-word } X,\ B^c \text{ accepts } X \ \leftrightarrow\ B \text{ does not accept } X''.$$

The above propositions hold because the standard proofs are directly formalisable in ACA_0. Complementation is the more interesting of the two cases since its proof involves nontrivial infinite combinatorics. For example, the crucial step in Büchi's original proof invokes the infinite Ramsey Theorem for pairs (see, e.g., [25]). This does not present a problem for the formalisation because the (general) infinite Ramsey Theorem holds in ACA_0 (see, e.g., [20]). A full analysis can be found in the recent paper [15], where it is shown that Proposition 12 is equivalent to the schema Σ^0_2-IND of Σ^0_2-induction, relative to the weak subsystem of second-order arithmetic RCA_0. In particular, this means that Proposition 12 is provable in $\mathsf{RCA}_0 + \Sigma^0_2$-IND, which is a subsystem of ACA_0.

If B is a Büchi automaton, we write $\rightarrow^*_B$ for the *reachability* relation on states of B (the transitive-reflexive closure of the label-erased transition relation $\rightarrow_B$). The ternary relation "B is a Büchi automaton and $y \rightarrow^*_B z$" is definable by a formula in second-order arithmetic with free first-order variables x, y, z where the parameter x supplies B via its code.

Proposition 13 (Formalised non-emptiness criterion). ACA_0 *proves: "*for every Büchi automaton B, there exists $X \in \Sigma^\omega$ accepted by B if and only if there exist states q_0, q_1 of B such that q_0 is initial, q_1 is accepting and $q_0 \rightarrow^*_B q_1 \rightarrow^*_B q_1$*"*.

We omit the proof, which is once again a straightforward formalisation of the (this time easy) standard argument.

Lemma 14. *If B is a Büchi automaton recognising the empty language then* ACA_0 *proves: "*for every infinite Σ-word X, the automaton B does not accept X*"*.

Proof. Suppose B recognises the empty language. By the standard non-formalised version of Proposition 13, there do not exist states q_0, q_1 of B such that q_0 is initial, q_1 is accepting and $q_0 \rightarrow^*_B q_1 \rightarrow^*_B q_1$. Since B is finite, the relation $\rightarrow^*_B$ can be encoded as a natural number $\ulcorner \rightarrow^*_B \urcorner$. Easily, ACA_0 proves: "$\ulcorner \rightarrow^*_B \urcorner$ encodes a transitive, reflexive relation R that contains $\rightarrow_B$, and there do not exist states q_0, q_1 of B such that q_0 is initial, q_1 is accepting and $q_0\, R\, q_1\, R\, q_1$". Since $\rightarrow^*_B$ is the smallest transitive-reflexive relation containing $\rightarrow_B$, it follows that ACA_0 proves: "there do not exist states q_0, q_1 of B such that q_0 is initial, q_1 is accepting and $q_0 \rightarrow^*_B q_1 \rightarrow^*_B q_1$". Hence, by Proposition 13, ACA_0 proves: "for every infinite Σ-word X, the automaton B does not accept X". □

Proof (of Lemma 9). Let G be a finite graph presentation of a regular pre-∞-proof. Consider the Büchi automata B_t and B_p defined in the proof of Theorem 5. By the definitions of B_t and B_p, the following statements are provable in ACA_0.

– "For all X, B_t accepts X if and only if X is a path through G that possesses an infinitely progressing trace."

– "For all X, B_p accepts X if and only if X is an infinite path through G."

Hence, by Propositions 11 and 12, ACA_0 proves:

$$\text{"unfold}(G) \text{ is an } \infty\text{-proof iff there is no } X \text{ such that } B_t^c \sqcap B_p \text{ accepts } X\text{"}. \quad (2)$$

Now, assume G presents an ∞-proof. We have, as in the proof of Theorem 5, $\mathcal{L}(B_t) \subseteq \mathcal{L}(B_p)$, hence the language of the Büchi automaton $B_t^c \sqcap B_p$ is empty. Hence, by Lemma 14, ACA_0 proves: "there is no X such that $B_t^c \sqcap B_p$ accepts X". It thus follows from (2) that ACA_0 proves: "unfold(G) is an ∞-proof", as required. □

7 Further Work

The following are natural possible continuations of the work in this paper.

(i) Give a syntactic rewrite-based proof eliminating non-atomic cuts from ∞-proofs. (This is done in [2,13] for different notions of cyclic proof.)
(ii) Give an explicit syntactic translation from cyclic proofs to proofs in PA. Is there an essential blow-up in size? (Because of Solovay's speed-up theorem for ACA_0 over PA, the indirect translation implicit in our proof has a potential non-elementary blow-up.)
(iii) Understand the power of cyclic proof in the context of weaker fragments of arithmetic. For example, what is the power of cyclic proof if restricted to Σ_1^0-sequents? The example in Fig. 2 shows that the resulting theory is stronger than $\mathsf{I}\Sigma_1$.
(iv) If the rules and axioms in Figs. 3, 4 and 5 are changed to their intuitionistic versions is the resulting notion of cyclic proof conservative over Heyting Arithmetic?
(v) Understand in general terms when cyclic proof is conservative over proof by induction (as in this paper), and when it is not (as in [3]).

References

1. Baelde, D.: On the proof theory of regular fixed points. In: Giese, M., Waaler, A. (eds.) TABLEAUX 2009. LNCS (LNAI), vol. 5607, pp. 93–107. Springer, Heidelberg (2009). doi:10.1007/978-3-642-02716-1_8
2. Baelde, D., Doumane, A., Saurin, A.: Infinitary proof theory: the multiplicative additive case. In: Talbot, J.-M., Regnier, L. (eds.) 25th EACSL Annual Conference on Computer Science Logic (CSL 2016). Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, vol. 62, pp. 42:1–42:17. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2016)
3. Berardi, S., Tatsuta, M.: Classical system of Martin-Löf's inductive definitions is not equivalent to cyclic proof system. In: Esparza, J., Murawski, A.S. (Eds.) FOSSACS 2017. LNCS, vol. 10203, pp. 301–317. Springer, Heidelberg (2017)
4. Brotherston, J.: Cyclic proofs for first-order logic with inductive definitions. In: Beckert, B. (ed.) TABLEAUX 2005. LNCS (LNAI), vol. 3702, pp. 78–92. Springer, Heidelberg (2005). doi:10.1007/11554554_8

5. Brotherston, J.: Sequent calculus proof systems for inductive definitions. Ph.D. thesis, University of Edinburgh, November 2006
6. Brotherston, J., Bornat, R., Calcagno, C.: Cyclic proofs of program termination in separation logic. In: Proceedings of POPL-35, pp. 101–112. ACM (2008)
7. Brotherston, J., Distefano, D., Petersen, R.L.: Automated cyclic entailment proofs in separation logic. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE 2011. LNCS (LNAI), vol. 6803, pp. 131–146. Springer, Heidelberg (2011). doi:10.1007/978-3-642-22438-6_12
8. Brotherston, J., Gorogiannis, N.: Cyclic abduction of inductively defined safety and termination preconditions. In: Müller-Olm, M., Seidl, H. (eds.) SAS 2014. LNCS, vol. 8723, pp. 68–84. Springer, Cham (2014). doi:10.1007/978-3-319-10936-7_5
9. Brotherston, J., Gorogiannis, N., Petersen, R.L.: A generic cyclic theorem prover. In: Jhala, R., Igarashi, A. (eds.) APLAS 2012. LNCS, vol. 7705, pp. 350–367. Springer, Heidelberg (2012). doi:10.1007/978-3-642-35182-2_25
10. Brotherston, J., Simpson, A.: Complete sequent calculi for induction and infinite descent. In: Proceedings of LICS-22, pp. 51–60. IEEE Computer Society, July 2007
11. Brotherston, J., Simpson, A.: Sequent calculi for induction and infinite descent. J. Logic Comput. **21**(6), 1177–1216 (2011)
12. Dax, C., Hofmann, M., Lange, M.: A proof system for the linear time μ-calculus. In: Arun-Kumar, S., Garg, N. (eds.) FSTTCS 2006. LNCS, vol. 4337, pp. 273–284. Springer, Heidelberg (2006). doi:10.1007/11944836_26
13. Fortier, J., Santocanale, L.: Cuts for circular proofs: semantics and cut-elimination. In: Computer Science Logic 2013 (CSL 2013), CSL 2013, 2–5 September 2013, Torino, Italy, pp. 248–262 (2013)
14. Kaye, R.: Models of Peano Arithmetic. Number 15 in Oxford Logic Guides. Oxford University Press, Oxford (1991)
15. Kolodziejczyk, L.A., Michalewski, H., Pradic, P., Skrzypczak, M.: The logical strength of Büchi's decidability theorem. In: Talbot, J.-M., Regnier, L. (eds.) 25th EACSL Annual Conference on Computer Science Logic (CSL 2016). Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, vol. 62, pp. 36:1–36:16. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2016)
16. Martin-Löf, P.: Haupstatz for the intuitionistic theory of iterated inductive def initions. In: Fenstad, J.E. (ed.) Proceedings of the Second Scandinavian Logic Symposium, pp. 179–216. North Holland (1971)
17. Mio, M., Simpson, A.: A proof system for compositional verification of probabilistic concurrent processes. In: Pfenning, F. (ed.) FoSSaCS 2013. LNCS, vol. 7794, pp. 161–176. Springer, Heidelberg (2013). doi:10.1007/978-3-642-37075-5_11
18. Niwiński, D., Walukiewicz, I.: Games for the μ-calculus. Theoret. Comput. Sci. **163**, 99–116 (1997)
19. Santocanale, L.: A calculus of circular proofs and its categorical semantics. In: Nielsen, M., Engberg, U. (eds.) FoSSaCS 2002. LNCS, vol. 2303, pp. 357–371. Springer, Heidelberg (2002). doi:10.1007/3-540-45931-6_25
20. Simpson, S.G.: Subsystems of Second Order Arithmetic. Perspectives in Logic. Association for Symbolic Logic, New York (2009)
21. Sprenger, C., Dam, M.: A note on global induction mechanisms in a μ-calculus with explicit approximations. Theor. Inform. Appl. **37**, 365–399 (2003)
22. Sprenger, C., Dam, M.: On the structure of inductive reasoning: circular and tree-shaped proofs in the μcalculus. In: Gordon, A.D. (ed.) FoSSaCS 2003. LNCS, vol. 2620, pp. 425–440. Springer, Heidelberg (2003). doi:10.1007/3-540-36576-1_27

23. Stratulat, S.: Structural vs. cyclic induction: a report on some experiments with Coq. In: SYNASC 2016: Proceedings of the 18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, pp. 27–34. IEEE Computer Society (2016)
24. Studer, T.: On the proof theory of the modal mu-calculus. Stud. Logica. **89**(3), 343–363 (2008)
25. Thomas, W.: Automata on infinite objects. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science. Formal Models and Semantics, vol. B, pp. 133–192. Elsevier Science Publishers, Amsterdam (1990)

Classical System of Martin-Löf's Inductive Definitions Is Not Equivalent to Cyclic Proof System

Stefano Berardi[1(✉)] and Makoto Tatsuta[2]

[1] Università di Torino, Turin, Italy
stefano@di.unito.it
[2] National Institute of Informatics/Sokendai, Tokyo, Japan

Abstract. A cyclic proof system, called $\texttt{CLKID}^\omega$, gives us another way of representing inductive definitions and efficient proof search. The 2011 paper by Brotherston and Simpson showed that the provability of $\texttt{CLKID}^\omega$ includes the provability of Martin-Löf's system of inductive definitions, called LKID, and conjectured the equivalence. Since then, the equivalence has been left an open question. This paper shows that $\texttt{CLKID}^\omega$ and LKID are indeed not equivalent. This paper considers a statement called 2-Hydra in these two systems with the first-order language formed by 0, the successor, the natural number predicate, and a binary predicate symbol used to express 2-Hydra. This paper shows that the 2-Hydra statement is provable in $\texttt{CLKID}^\omega$, but the statement is not provable in LKID, by constructing some Henkin model where the statement is false.

1 Introduction

An inductive definition is a way to define a predicate by an expression which may contain the predicate itself. The predicate is interpreted by the least fixed point of the defining equation. Inductive definitions are important in computer science, since they can define useful recursive data structures such as lists and trees. Inductive definitions are important also in mathematical logic, since they increase the proof theoretic strength. Martin-Löf's system of inductive definitions given in [10] is one of the most popular system of inductive definitions. This system has production rules for an inductive predicate, and the production rule determines the introduction rule and the elimination rule for the predicate.

Brotherston [3] and Simpson [6] proposed an alternative formalization of inductive definitions, called a cyclic proof system. A proof, called *a cyclic proof*, is defined by proof search, going upwardly in a proof figure. If we encounter the same sequent (called a bud) as some sequent we already passed (called a companion) we can stop. The induction rule is replaced by a case rule, for this purpose. The soundness is guaranteed by some additional condition, called the *global trace condition*, which guarantees the case rule decreases some measure of a bud from that of the companion. In general, for proof search, a cyclic proof system can find an induction formula in a more efficient way than Martin-Löf's

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 301–317, 2017.
DOI: 10.1007/978-3-662-54458-7_18

system, since a cyclic proof system does not have to choose fixed induction formula in advance. A cyclic proof system enables us efficient implementation of theorem provers with inductive definitions [2,4,5,7]. In particular, it works well for theorem provers of separation logic.

Brotherston and Simpson [6] investigated Martin-Löf's system LKID of inductive definitions in classical logic for the first-order language, and the cyclic proof system $\mathtt{CLKID}^\omega$ for the same language, showed the provability of $\mathtt{CLKID}^\omega$ includes that of LKID, and conjectured the equivalence. Since then, the equivalence has been left an open question. Simpson [11] submitted a proof of a particular case of the conjecture, for the theory of Peano Arithmetic.

This paper shows $\mathtt{CLKID}^\omega$ and LKID are indeed not equivalent. To this aim, we will consider the first-order language formed by 0, the successor s, the natural number predicate N, and a binary predicate symbol p. We introduce a statement we call 2-Hydra, which is a miniature version of the Hydra problem considered by Laurence Kirby and Jeff Paris [9]: the proviso "2" means that we only have two "heads". We define some statement, called the 2-Hydra statement, and we show that the 2-Hydra statement is provable in $\mathtt{CLKID}^\omega$ with the language, but the statement is not provable in LKID with the language. 2-Hydra is similar to a candidate for a counter-example proposed by S. Stratulat [12]. The unprovability is shown by constructing some model of $\mathtt{CLKID}^\omega$ where 2-Hydra is false.

For constructing the counter model $\mathcal{M}$ for the second result, we take both the universe of $\mathcal{M}$ and the interpretation of the predicate N to be $\mathtt{Nat}+\mathtt{Z}$, where Nat is the set of natural numbers and Z is the set of integers, and some predicate p which is a counter-example of 2-Hydra. We prove that $\mathcal{M}$ is a model of LKID by using a set of partial bijections on $\mathcal{M}$ and a quantifier elimination result.

The quantifier elimination theorem for a theory of partial equivalence relations is new, as far as we know, and it may have some independent interest.

This model also shows that LKID is not conservative when we add inductive predicates, namely, it is not the case that for any language L, the system of LKID with language L and any additional inductive predicate is conservative over the system of LKID with L.

Section 2 describes Brotherston-Simpson conjecture. Section 3 defines the 2-Hydra statement and proves the 2-Hydra statement in $\mathtt{CLKID}^\omega$. Section 4 defines the counter model $\mathcal{M}$ and the proof outline of it. Section 5 introduces a family of partial bijections. Section 6 proves a quantifier elimination theorem for a theory of partial bijections. Section 7 proves that the 2-Hydra statement is not provable in LKID. Section 8 shows non-conservativity of LKID with additional inductive predicates. We conclude in Sect. 9. Detailed proofs are in [1].

2 Brotherston-Simpson Conjecture

In this section we introduce Brotherston-Simpson Conjecture.

2.1 Martin-Löf's Inductive Definition System LKID

We briefly remind you of Martin-Löf's inductive definition system LKID, defined in detail in [6].

The language of LKID is determined by a first-order language with inductive predicate symbols. The logical system LKID is determined by production rules for inductive predicate symbols. These production rules mean that the inductive predicate denotes the least fixed point defined by these production rules.

We often abbreviate $p(t), q(t,u)$ with pt, qtu. For example, for an inductive predicate symbol N, the production rules may be written as

$$\frac{}{N0} \qquad \frac{Nx}{Nsx}$$

These production rules mean that N denotes the smallest set closed under 0 and s, namely the set of natural numbers. We call this set of production rules Φ_N.

The inference rules of LKID are standard inference rules in classical first-order logic LK with the introduction rules and the elimination rules for inductive predicates, determined by the production rules. These rules describe that the predicate actually denotes the least fixed point. In particular, the elimination rule describes the induction principle.

For example, the above production rules give the introduction rules

$$\frac{}{\Gamma \vdash N0, \Delta} \qquad \frac{\Gamma \vdash Nx, \Delta}{\Gamma \vdash Nsx, \Delta}$$

and the elimination rule

$$\frac{\Gamma \vdash F0, \Delta \quad \Gamma, Fx \vdash Fsx, \Delta \quad \Gamma, Ft \vdash \Delta}{\Gamma, Nt \vdash \Delta}$$

This elimination rule describes mathematical induction principle restricted to N. LKID is sound with respect to a class of models called Henkin models (Definition 2.10 of [6]). We omit the definition of Henkin models and we only use the following property: if a first order structure $\mathcal{M}$ satisfies the induction schema for N, then $\mathcal{M}$ is an Henkin model of LKID with the predicate N.

2.2 Cyclic Proof System CLKID$^\omega$

A cyclic proof system CLKID$^\omega$ [6] is defined as a system obtained from LKID by (1) replacing elimination rules by case rules, (2) allowing a bud as an open assumption and requiring a companion for each bud, (3) requiring the global trace condition.

The case rule is defined by unfolding the production rule in the antecedent. For example, the case rule for N is

$$\frac{\Gamma, t = 0 \vdash \Delta \quad \Gamma, t = sx, Nx \vdash \Delta}{\Gamma, Nt \vdash \Delta}$$

In a cyclic proof, we can have open assumptions, called *buds*, but it is required that each bud has some corresponding sequent of the same form, called a *companion*, inside the proof figure.

An example of a cyclic proof is

$$\dfrac{\dfrac{\dfrac{}{\vdash N0}}{\vdash Ns0}}{\vdash Nss0}\quad\dfrac{\dfrac{(a)Nx \vdash Nssx}{Nx' \vdash Nssx'}\ (\text{subst})}{Nx' \vdash Nsssx'}\ (case)}{(a)Nx \vdash Nssx}$$

where the mark (a) denotes the bud-companion pair. Remark that the companion (a) uses Nx, but the bud (a) uses Nx where x is x', so their actual meanings are different even though they are of the same form.

A *pre-proof* of $\mathtt{CLKID}^\omega$ is obtained by recursively replacing every bud by the proof of its companion. A *trace* is a sequence of occurrences of an atom in a path of the proof tree, possibly moving to a case-descendant when passing through a case rule. Moving to a case-descendant is called a *progress point* of the trace (Definition 5.4 [6]). The *global trace condition* says that for every infinite path there is a trace with infinitely many progress points following some tail of the path (Definition 5.5 [6]). The global trace condition guarantees the soundness of a cyclic proof system for fixed-point models. $\mathtt{CLKID}^\omega$ is not known to be sound for Henkin models, and this leaves the possibility of having an Henkin counter-model for a theorem of $\mathtt{CLKID}^\omega$.

2.3 Brotherston-Simpson Conjecture

LKID has been often used for formalizing inductive definitions, while $\mathtt{CLKID}^\omega$ is another way for formalizing inductive definitions, and moreover $\mathtt{CLKID}^\omega$ is more suitable for proof search. This raises the question of the relationship between LKID and cyclic proofs: Brotherston and Simpson conjectured the equality for each inductive definition. The left-to-right inclusion is proved in [3], Lemma 7.3.1 and in [6], Theorem 7.6. Brotherston-Simpson conjecture (the Conjecture 7.7 in [6]) is that the provability LKID includes that of $\mathtt{CLKID}^\omega$. Simpson [11] submitted a proof of the conjecture in the case of Peano Arithmetic. The goal of this paper is to prove that it is false in general.

3 2-Hydra Problem

3.1 Hydra Problem

The Hydra of Lerna was a mythological monster, popping two smaller heads whenever you cut one. It was a swamp creature (its name means "water") and possibly was the swamp itself, whose heads are the swamp plants, with two smaller plants growing whenever you cut one. The original Hydra was defeated

by fire, preventing heads to grow again. In the mathematical problem of Hydra, we ask whether we may destroy an Hydra just by cutting heads.

Laurence Kirby and Jeff Paris [9] formulated the Hydra problem as a statement for mathematical trees. We are interested about making Hydra a problem for natural numbers, representing the length of a head, and restricting to the case when the number of heads is always 2. We call our statement 2-Hydra.

3.2 2-Hydra Statement

In this subsection we give the 2-Hydra statement, which is a formula saying that any 2-hydra eventually looses its two heads. This statement actually will give a counterexample to Brotherston-Simpson conjecture.

Let Σ_N be the signature $\{0, s, N, p\}$ of a first order language, where 0, the successor s, an inductive predicate N for natural numbers, and an ordinary binary predicate symbol p. The logical system $\mathtt{LKID}(\Sigma_N, \Phi_N)$ is defined as the system LKID with the signature Σ_N and the production rules Φ_N.

We consider a formal statement of 2-Hydra. The number of head is always 2. Either both heads have positive length, you reduce the length of the first head by 1 unit, and of the second head by 2 units (if possible), or there is a unique head with positive length, you duplicate it and you reduce it by 1 and by 2 units (if possible). We may express H by the convergence of the following set of transformations on $n, m \in \mathtt{Nat}$: if $n \geq 1$ and $m \geq 2$ then $(n, m) \mapsto (n-1, m-2)$; if $n \geq 2$ then $(n, 0) \mapsto (n-1, n-2)$; if $m \geq 2$ then $(0, m) \mapsto (m-1, m-2)$. When no transformation applies we stop. We may define H by a formula in the language Σ_N: the intended meaning of p is the complement of the union of all infinite sequences of transformations. From now on, we write $A_1, \ldots, A_n \to B$ for $A_1 \wedge \ldots \wedge A_n \to C$ and $\forall x_1, \ldots, x_n \in N.\ A$ for $\forall x_1.\ \ldots.\ \forall x_n.\ N(x_1) \wedge \ldots \wedge N(x_n) \to A$.

Definition 1 (2-Hydra Statement H). *We define* $H = (H_a, H_b, H_c, H_d \to \forall x, y \in N.\ p(x, y))$, *where* H_a, H_b, H_c, H_d *are:*

(H_a) $\forall x \in N.\ p(0,0) \wedge p(s0,0) \wedge p(x, s0)$,
(H_b) $\forall x, y \in N.\ p(x, y) \to p(sx, ssy)$,
(H_c) $\forall y \in N.\ p(sy, y) \to p(0, ssy)$,
(H_d) $\forall x \in N.\ p(sx, x) \to p(ssx, 0)$.

For all $n, m \in \mathtt{Nat}$ there is a unique formula among H_a, H_b, H_c, H_d having some instance inferring $p(n, m)$. The assumption $p(n', m')$ of such a formula, if any assumption exists, satisfies $\max(n', m') < \max(n, m)$. Thus, we may prove H in PA by induction on $\max(n, m)$. We could define p as an inductive predicate: however, we preferred having p just a predicate symbol, because in this way the definition of a counter-model does not require to check the inductive rule for p.

We will prove that $\mathtt{LKID}(\Sigma_N, \Phi_N) + (0, s)$-axioms does not prove 2-Hydra. We define the $(0, s)$-axioms as the axioms "0 is not successor" or $\forall x \in N.\ sx \neq 0$, and "successor is injective", or $\forall x, y \in N.\ sx = sy \to x = y$. These axioms

cannot be proved in $\mathtt{LKID}(\Sigma_N, \Phi_N)$, because they fail, respectively, in the model of $\mathtt{LKID}(\Sigma_N, \Phi_N)$ uniquely determined by $\mathcal{M} = N_{\mathcal{M}} = \{0\}$, $s0 = 0$, in the model uniquely determined by $\mathcal{M} = N_{\mathcal{M}} = \{0, s0\}$, $0 \neq s0$ and $ss0 = s0$. Compared with PA, in $\mathtt{LKID}(\Sigma_N, \Phi_N) + (0, s)$-axioms we do not have a sum nor a product on N, nor we have inductive predicate symbols for addition or multiplication.

3.3 2-Hydra Statement in Cyclic-Proof System

In this section, we give a cyclic proof of the 2-Hydra statement.

We define the logical system $\mathtt{CLKID}^\omega(\Sigma_N, \Phi_N)$ as the system $\mathtt{CLKID}^\omega$ with the signature Σ_N and the production rules Φ_N.

Theorem 1. *The 2-Hydra statement H is provable in $\mathtt{CLKID}^\omega(\Sigma_N, \Phi_N)$.*

Proof. Let the 2-Hydra axioms $\hat{H}$ be H_a, H_b, H_c, H_d defined in Definition 1.

For simplicity, we will write the use of 2-Hydra axioms by omitting (Cut), $(\to R)$, $(\forall\ R)$, (Axiom), as in the following example.

$$\frac{\hat{H}, Nsy'', Ny'' \vdash psy''y''}{\hat{H}, Nsy'', Ny'' \vdash p0ssy''}$$

We will also write a combination of (Case) and (=L) as one rule in the following example.

$$\frac{\hat{H} \vdash p00 \quad \hat{H}, Nx' \vdash psx'0}{\hat{H}, Nx \vdash px0}\ Nx$$

For saving space, we omit writing $\hat{H}$ in every sequent int the next proof figure. For example, $Nx, Ny \vdash pxy$ actually denotes $\hat{H}, Nx, Ny \vdash pxy$.

The following is a cyclic proof of $Nx, Ny \vdash pxy$.

$$\dfrac{\dfrac{\dfrac{}{\vdash p00} \quad \dfrac{\dfrac{\dfrac{(a)Nx, Ny \vdash pxy}{Nsx'', Nx'' \vdash psx''x''}}{Nsx'', Nx'' \vdash pssx''0}}{Nx' \vdash psx'0}\ Nx}{Nx \vdash px0} \quad \dfrac{\dfrac{}{N0, Nx \vdash px1} \quad \dfrac{\dfrac{\dfrac{(a)Nx, Ny \vdash pxy}{Nsy'', Ny'' \vdash psy''y''}}{Nsy'', Ny'' \vdash p0ssy''} \quad \dfrac{\dfrac{(a)Nx, Ny \vdash pxy}{Nx', Ny'' \vdash px'y''}}{Nsy'', Nx', Ny'' \vdash psx'ssy''}}{Nsy'', Nx, Ny'' \vdash pxssy''}\ Nx}{Nx, Ny' \vdash pxsy'}\ Ny}{(a)Nx, Ny \vdash pxy}$$

The global trace condition holds for the following reason (the detailed proof is in [1]). We have three possible choices for constructing an infinite path in the proof: taking a bud of the left, middle, or right. For a given bud and $z_1, z_2 \in \{x, y\}$, we write $z_1 \rightsquigarrow z_2$ for a progressing trace from Nz_1 in the companion to Nz_2 in the bud. We write $z_1 \rightsquigarrow z_2, z_3$ for $z_1 \rightsquigarrow z_2$ and $z_1 \rightsquigarrow z_3$. For the left bud, there are $x \rightsquigarrow x, y$. For the middle bud, there are $y \rightsquigarrow x, y$. For the left bud, there are $x \rightsquigarrow x$ and $y \rightsquigarrow y$. Hence, given an infinite path, there is some tail of the path with an infinitely progressing trace, by cleverly choosing x and y possibly alternatively. Hence the global trace condition holds. □

4 A Structure $\mathcal{M}$ for the Language Σ_N Falsifying 2-Hydra

In this section we define a structure $\mathcal{M}$ for the language Σ_N, we prove that $\mathcal{M}$ falsifies the 2-Hydra statement H, and we characterize the subsets of $\mathcal{M}$ which satisfy the induction schema.

4.1 Outline of Proof of Non-Provability

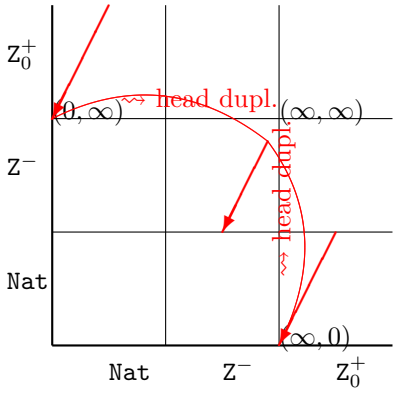
In Sect. 4, we define a counter model $\mathcal{M}$. The most difficult point is to prove that $\mathcal{M}$ satisfies Definition 2.10 of [6] for having an Henkin model of $\mathtt{LKID} + \Sigma_N$. We will prove a sufficient condition for it, the induction schema for N.

On one hand, we prove that in our structure $\mathcal{M}$ all unary definable predicates of $\mathcal{M}$ are sets whose measure is some dyadic rational number. This involves proving a quantifier-elimination result (Sect. 6) for a theory of partial equivalence relations (Sect. 5). This result is new, as far as we know: for an introduction to quantifier-elimination we refer to [8], Sects. 3.1 and 3.2). On the other hand, Sect. 4.3 shows that a definable set of $\mathcal{M}$ with dyadic measure satisfies the induction schema. Combining them, finally we will show that $\mathcal{M}$ satisfies the induction schema for N and according to Definition 2.10 of [6] is an Henkin model of $\mathtt{LKID} + \Sigma_N$.

4.2 Definition of the Structure $\mathcal{M}$

Let $\mathtt{Z}$ be the set of relative integers. $\mathcal{M}$ is $\mathtt{Nat} + \mathtt{Z}$: we represent $\mathtt{Nat} + \mathtt{Z}$ by $\{(1, x) \mid x \in \mathtt{Nat}\} \cup \{(2, x) \mid x \in \mathtt{Z}\}$. We first define the interpretations $0_{\mathcal{M}}, s_{\mathcal{M}}, N_{\mathcal{M}}$. $0_{\mathcal{M}}$ is 0 in the component $\mathtt{Nat}$ and $s_{\mathcal{M}}$ is the successor on $\mathtt{Nat}$ and on $\mathtt{Z}$. We choose $N_{\mathcal{M}} = \mathcal{M}$: by construction, $\mathcal{M}$ satisfies the $(0, s)$-axioms. We abbreviate $x + n = s^n_{\mathcal{M}}(x)$, ∞ for the 0 in the component $\mathtt{Z}$, and $\infty - n$ for the relative integer $-n$ in the component $\mathtt{Z}$, for all $n \in \mathtt{Nat}$. We define the following subsets of $\mathcal{M}$: $\underline{\mathtt{Nat}} = \{0_{\mathcal{M}} + n | n \in \mathtt{Nat}\}$ and $\underline{\mathtt{Z}}^- = \{\infty - (n + 1) | n \in \mathtt{Nat}\}$ and $\underline{\mathtt{Z}}^+_0 = \{\infty + n | n \in \mathtt{Nat}\}$. The sets $\underline{\mathtt{Nat}}$, $\underline{\mathtt{Z}}^-$, $\underline{\mathtt{Z}}^+_0$ are a partition of $\mathcal{M}$.

In order to complete the definition of $\mathcal{M}$ we have to choose the interpretation $p_{\mathcal{M}}$ of the binary predicate p. We first define $\pi = \{(n, 2n) | n \in \mathtt{Nat}\} \subseteq \mathtt{Nat} \times \mathtt{Nat}$. π is the set of points of the straight line $y = 2x$ whose coordinates are some pair of natural numbers. We imagine π starting from the infinite, moving at each step from some (sa, ssb) to (a, b), and ending in $(0, 0)$. Given any $(m_1, m_2) \in \mathcal{M} \times \mathcal{M}$ we define $(m_1, m_2) + \pi = \{(m_1 + a, m_2 + b) | (a, b) \in \pi\}$ and $(m_1, m_2) - \pi = \{(m_1 - a, m_2 - b) | (a, b) \in \pi\}$. We define three paths in $\mathcal{M} \times \mathcal{M}$ by $\pi_1 = (0_{\mathcal{M}}, \infty) + \pi$ and $\pi_2 = (\infty, 0_{\mathcal{M}}) + \pi$ and $\pi_3 = (\infty - 1, \infty - 2) - \pi$. Eventually, we set $p_{\mathcal{M}} = \mathcal{M}^2 \setminus (\pi_1 \cup \pi_2 \cup \pi_3)$. As explained in the figure below, we may move forever along $\pi_1 \cup \pi_2 \cup \pi_3$ (in red) while "cutting heads" as follows: $\ldots \mapsto (0_{\mathcal{M}} + 2, \infty + 4) \mapsto (0_{\mathcal{M}} + 1, \infty + 2) \mapsto (0_{\mathcal{M}}, \infty) \mapsto (\infty - 1, \infty - 2) \mapsto (\infty - 2, \infty - 4) \mapsto \ldots$.



Lemma 1 (The 2-Hydra Lemma). *H is false in $\mathcal{M}$*

$\mathcal{M}$ satisfies by construction the closure of N under 0 and s, and the $(0,s)$-axioms. In order to prove that $\mathcal{M}$ is a model for $\mathtt{LKID}(\Sigma_N,\Phi_N)+(0,s)$-axioms, we have to prove that $\mathcal{M}$ satisfies Definition 2.10 of [6], Definition 2.10). Let $\mathcal{H}$ be the set of definable predicates of $\mathcal{M}$. A predicate $P\subseteq\mathcal{M}^n$ is definable in $\mathcal{M}$ if for some formula A in the language Σ_N plus constants denoting the elements of $\mathcal{M}$, and with free variables in $x_1,\ldots,x_n$, we have $P=\{(m_1,\ldots,m_n)\in\mathcal{M}^n|\mathcal{M}\models A[m_1/x_1,\ldots,m_n/x_n]\}$. We write $\mathcal{H}_n$ for the subset of definable predicates of arity n: we call $\mathcal{H}_1$ the set of definable *sets* of $\mathcal{M}$. According to Definition 2.10 of [6], we have to prove that $\mathcal{M}$ is the smallest pre-fixed point in $\mathcal{H}_1$ for the inductive definition of N: a sufficient condition is to prove the induction schema, that all $X\in\mathcal{H}_1$ which are closed under 0 and s are equal to $\mathcal{M}$.

4.3 The Measure of the Subsets of $\mathcal{M}$ Closed Under 0 and s

In this subsection we define a sufficient condition for a predicate on $\mathcal{M}$ to satisfy the induction schema, by using a finitely additive measure $\mu(X)$, defined on some subsets $X\subseteq\mathcal{M}$. We will prove that all definable subsets of $\mathcal{M}$ satisfy this condition.

Definition 2 (Measure of a Subset of $\mathcal{M}$). *For any $X\subseteq\mathcal{M}$ we set: $\mu(X)=\lim_{x\to\infty}\frac{\mathtt{card}(\{0_{\mathcal{M}}+n,\infty-n,\infty+n\in\mathcal{M}|n\in[0,x]\}\cap X)}{3(x+1)}$ whenever this limit exists.*

For instance, $\mu(\underline{\mathtt{Nat}})=1/3$ and if $E=\{0_{\mathcal{M}},0_{\mathcal{M}}+2,\ldots,\infty-2,\infty,\infty+2,\ldots\}$, then $\mu(E)=1/2$. We may now provide a sufficient condition for a predicate to satisfy the induction rule.

Lemma 2 (Measure Lemma). *If $\mu(P)$ is a dyadic rational, then P satisfies the induction schema.*

An example: if $P=\underline{\mathtt{Nat}}\cup\underline{\mathtt{Z}}_0^+$, then P is closed under 0, s and $\infty-1\notin P$. P does *not* satisfy the induction schema and $\mu(P)=2/3$ is *not* dyadic.

5 A Set $\mathcal{R}$ of Partial Bijections on $\mathcal{M}$

In this section we introduce some set $\mathcal{R}$ of partial bijections on $\mathcal{M}$, whose domain have measure some dyadic rational. In Sects. 6 and 7 we will prove that all definable predicates in $\mathcal{M}$ are a boolean combination of atomic formulas of the language $\mathcal{R}$, and that all definable sets in $\mathcal{M}$ are domains of bijections in $\mathcal{R}$, therefore all have measure some dyadic rational, and by Lemma 2 satisfy the induction schema. We will conclude that $\mathcal{M}$ is an Henkin model of $\mathtt{LKID} + \Sigma_N$.

We say that a relation R is finite if there are finitely many pairs $(x, y) \in R$. For any set X and any binary relations R, S we write: $\mathtt{id}_X = \{(x, x) | x \in X\}$, $\mathtt{dom}(R) = \{x | \exists y.(x, y) \in R\}$, $\mathtt{codom}(R) = \{y | \exists x.(x, y) \in R\}$, $R^{-1} = \{(y, x) | (x, y) \in R\}$, $R \circ S = \{(x, z) | \exists y.((x, y) \in S) \wedge ((y, z) \in R)\}$ and $R \lceil X = \{(x, y) \in R | x \in X\}$. Remark that we defined relation composition *in the same order as function composition*: the reason is that we will only consider relations which are partial bijections.

5.1 The Set $\mathcal{D}$ of Subsets of $\mathcal{M}$

In this subsection we propose a candidate $\mathcal{D}$ for the definable subsets of $\mathcal{M}$.

For any sets I, J we define $I \subseteq_\sim J$ as "$(I \setminus J)$ is finite": this means "$I \subseteq J$ up to finitely many elements. We define $I \sim J$ as $I \subseteq_\sim J \wedge J \subseteq_\sim I$: this means "$I, J$ are equal up to finitely many elements". $I \sim J$ is equivalent to: $(I \setminus J) \cup (J \setminus I)$ is finite.

For any $r, s \in \mathtt{Q}$ we introduce the formal notations $0_\mathcal{M} + r$, $\infty + s$: they denote elements of $\mathcal{M}$ if and only if $r \in \mathtt{Nat}$, $s \in \mathtt{Z}$. For any $z \in \mathtt{Z}$, $r \in \mathtt{Q}$, we define the set of formal notations $M(2^z, r) = \{0_\mathcal{M} + (2^z * n + r), \infty + (2^z * w + r) | (n \in \mathtt{Nat}) \wedge (w \in \mathtt{Z})\}$. We denote with $\mathcal{B}$ the set of all sets $M(2^z, r)$, for some $z \in \mathtt{Z}$, $r \in \mathtt{Q}$.

We define $\mathcal{D}$ as the family of subsets of $\mathcal{M}$ which are equivalent, up to finitely many elements, to some finite union of sets in $\mathcal{B}$.

Definition 3 (The Family $\mathcal{D}$). *$D \in \mathcal{D}$ if and only if $D \sim (B_1 \cup \ldots \cup B_n)$ for some $B_1, \ldots, B_n \in \mathcal{B}$. We call $\mathcal{D}$ the dyadic family.*

Since $2^z > 0$, all sets $M(2^z, r)$ are infinite. We have $M(2^z, r) \subseteq_\sim \mathcal{M}$ if and only if $(2^z * n + r) \in \mathtt{Nat}$ for all but finitely many $n \in \mathtt{Nat}$ and $(2^z * w + r) \in \mathtt{Z}$ for all but finitely many $w \in \mathtt{Z}$. We may check that this is equivalent to: $z \in \mathtt{Nat}$ and $r \in \mathtt{Z}$.

We prove that every set in $\mathcal{D}$ has measure some dyadic rational.

Lemma 3 ($\mathcal{D}$-Lemma). *Let $a_0, a \in \mathtt{Z}$ and $D \in \mathcal{D}$.*

1. *All finite subsets of $\mathcal{M}$ are in $\mathcal{D}$.*
2. *For all $a \geq a_0$ there are $0 \leq b_1 < \ldots < b_i < 2^a$ such that $M(2^{a_0}, b) = (M(2^a, b_1) \cup \ldots \cup M(2^a, b_i))$.*
3. *For some a_0 and for all $a \geq a_0$ there are $0 \leq b_1, \ldots, b_i < 2^a$ such that $D \sim (M(2^a, b_1) \cup \ldots \cup M(2^a, b_i))$.*

4. $\mu(D)$ *is some dyadic rational.*
5. $\mathcal{D}$ *is closed under* $\sim$ *and all boolean operations.*

5.2 The Family $\mathcal{R}$ of Partial Bijections on $\mathcal{M}$

In this subsection we define a family $\mathcal{R}$ of partial bijections on $\mathcal{M}$ with domain in $\mathcal{D}$. The elements of $\mathcal{R}$ up to finitely many elements are empty or are some power of the complement of $p_{\mathcal{M}}$, restricted to some $D \in \mathcal{D}$.

We define first some set $\mathcal{F}$ of straight lines. $\mathcal{F}$ is the set of maps $\phi : \mathtt{Q} \to \mathtt{Q}$, defined by $\phi(x) = 2^z x + r$ for some $z \in \mathtt{Z}$ and some $r \in \mathtt{Q}$. $\mathcal{F}$ is closed under inverse: if $\phi(x) = 2^z x + r$, then $\phi^{-1}(x) = 2^{-z}x - r/2^z$. $\mathcal{F}$ is closed under composition: if $\phi_i(x) = 2^{z_i}x + r_i$ for $i = 1, 2$, then $\phi_2(\phi_1(x)) = 2^{z_1+z_2}x + (2^{z_2}r_1 + r_2)$.

Let $\mathtt{Q}+\mathtt{Q} = \{(i, r) | i = 1, 2 \wedge r \in \mathtt{Q}\}$. We extend $\phi : \mathtt{Q} \to \mathtt{Q}$ to a map $: \mathcal{M} \to \mathtt{Q}+\mathtt{Q}$ by $\phi((i, r)) = (i, \phi(r))$ (recall that each element of $\mathcal{M}$ is coded by some pair (i, r)). For any $D \subseteq \mathcal{M}$ we define $\phi(D) = \{\phi(d) | d \in D\} \subseteq \mathtt{Q} + \mathtt{Q}$. We provide a sufficient condition for having $\phi(D) \subseteq \mathcal{M}$ and $\phi(D) \in \mathcal{D}$.

Lemma 4 (ϕ-Lemma). *Let* $\phi(x) = 2^{z_1}x + r_1$ *for some* $z_1 \in \mathtt{Z}$ *and some* $r_1 \in \mathtt{Q}$ *and all* $x \in \mathtt{Q}$. *Assume* $M(2^z, r) \in \mathcal{B}$.

1. $\phi(M(2^z, r)) \in \mathcal{B}$.
2. If $D \in \mathcal{D}$ *and* $\phi(D) \overset{\subseteq}{\sim} \mathcal{M}$ *then* $\phi(D) \in \mathcal{D}$.

Proof.

1. We have $\phi(M(2^z, r)) = M(2^{z+z_1}, 2^{z_1}r + r_1) \in \mathcal{B}$.
2. If $D \in \mathcal{D}$ then $D \sim M(2^{a_1}, b_1) \cup \ldots \cup M(2^{a_i}, b_i)$ for some $a_1, \ldots, a_i \in \mathtt{Z}$ and some $b_1, \ldots, b_i \in \mathtt{Q}$. Then $\phi(D) \sim \phi(M(2^{a_1}, b_1)) \cup \ldots \cup \phi(M(2^{a_i}, b_i))$, and by point 1 above $\phi(M(2^{a_1}, b_1)) \cup \ldots \cup \phi(M(2^{a_i}, b_i)) \in \mathcal{B}$. Thus, $\phi(D) \in \mathcal{D}$.

A *partial bijection* on $\mathcal{M}$ is a bijection between two subsets of $\mathcal{M}$. We now define a family $\mathcal{R}$ of partial bijections on $\mathcal{M}$. For instance, one bijection in $\mathcal{R}$ is defined by $\phi(x) = 4x$, with domain $\mathcal{M}$ and codomain $M(4, 0)$, mapping $0_{\mathcal{M}} + n \mapsto 0_{\mathcal{M}} + 4n$ and $\infty + z \mapsto \infty + 4z$.

Let $\phi \in \mathcal{F}$, $\phi(x) = 2^z x + r$ with $z \in \mathtt{Z}$ and $r \in \mathtt{Q}$. We say that ϕ is *even* if z even, and that ϕ is *odd* if z is odd. We divide infinite bijections in $\mathcal{R}$ between "even" and "odd". They will be restrictions of an even or odd power of the relation $Q = \mathcal{M}^2 \setminus p_{\mathcal{M}}$, up to finitely many point. We will prove that the first order definable predicates of $\mathcal{M}$ are the propositional formulas of $\mathcal{R}$.

Definition 4 (Even Bijections). *Let* $D, E \in \mathcal{D}$ *and* $\phi \in \mathcal{F}$ *be even.* R *is an even* (D, E, ϕ)*-bijection if* D, E *are infinite,* R *is a bijection between* D, E, *and* R *is equal up to finitely many elements to the graph of* ϕ *restricted to* D, E: $R \sim \{(x, y) \in D \times E | y = \phi(x)\}$. *We denote the set of even bijections with* $\mathcal{R}^+$.

Q is a partial bijection on $\mathcal{M}$, and by definition Q maps $0_{\mathcal{M}} + n \mapsto \infty + 2n$ and $\infty + n \mapsto 0_{\mathcal{M}} + 2n$, and Q is associated to the odd map $\phi(x) = 2x$.

Definition 5 (Odd Bijections). *Let $\phi \in \mathcal{F}$ be odd. An odd (D, E, ϕ)-bijection is any bijection R between some infinite $D, E \in \mathcal{D}$, such that, up to finitely many points, R maps: (1) $\infty - n - 1 \mapsto \infty + \phi(-n-1)$, (2) $0_{\mathcal{M}} + n \mapsto \infty + \phi(n)$ and $\infty + n \mapsto 0_{\mathcal{M}} + \phi(n)$. We denote the set of odd bijections with $\mathcal{R}^-$.*

Q is an example of odd bijection. Let $\phi \in \mathcal{F}$ be even and $D, E \in \mathcal{D}$. A (D, E, ϕ)-even bijection may alternatively be defined as any bijection between D, E such that, up to finitely many points: (1) $\infty - n - 1 \mapsto \infty + \phi(-n-1)$, (2) $0_{\mathcal{M}} + n \mapsto 0_{\mathcal{M}} + \phi(n)$ and $\infty + n \mapsto \infty + \phi(n)$.

We define $\mathcal{R}_0$ as the set of all bijections between finite sets $D, E \in \mathcal{D}$. Eventually, we define a family $\mathcal{R}$ of partial bijections by $\mathcal{R} = \mathcal{R}^+ \cup \mathcal{R}_0 \cup \mathcal{R}^-$.

If $R \in \mathcal{R}^+ \cup \mathcal{R}^-$, associated to the map $\phi \in \mathcal{F}$ with domain D and codomain E, then we may prove that $E \sim \phi(D)$. $\mathcal{R}$ and $\mathcal{D}$ satisfy the following closure properties.

Lemma 5 (Partial bijections). *Assume that $R, S \in \mathcal{R}$ and $D \in \mathcal{D}$.*

1. $\mathtt{id}_D \in \mathcal{R}$
2. *If $D \in \mathcal{D}$ then $R(D) \in \mathcal{D}$*
3. $R \circ S \in \mathcal{R}$.
4. $R^{-1} \in \mathcal{R}$
5. *$\mathcal{D}$ is closed under complement.*

$\mathcal{R}$ is closed under intersection.

Lemma 6 (Closure Under Intersection). *Assume that $R, S \in \mathcal{R}$ are associated to $\phi, \psi \in \mathcal{F}$.*

1. *If $\phi = \psi$ then $R \cap S \in \mathcal{R}$*
2. *If $\phi \neq \psi$ then $R \cap S \in \mathcal{R}$*
3. *$\mathcal{R}$ is closed under intersections.*
4. *For all $R \in \mathcal{R}$ there is some $D \in \mathcal{D}$ such that $R \cap \mathtt{id}_{\mathcal{M}} = \mathtt{id}_D$.*

Our goal is to prove that every first-order definable subset of $\mathcal{M}$ is in $\mathcal{D}$. Since the sets definable in the language of $\mathcal{R}$ include those definable in $\mathcal{M}$, it is enough to prove that any first-order definable set in language of $\mathcal{R}$ is in $\mathcal{D}$. To this aim, we need a quantifier-elimination result for the language of $\mathcal{R}$.

6 A Quantifier Elimination Result for Partial Bijections

In this section we prove a quantifier elimination result for a theory of partial equivalence relation, which is the abstract counterpart of the families $\mathcal{R}$ and $\mathcal{D}$ introduced in the previous section. This is a simple, self-contained result introducing a model-theoretical tool of some interest.

Theorem 2 (Quantifier Elimination for Partial Bijections). *Let U be a set and $\mathcal{R}$ a set of partial bijections on U. Assume that all finite partial bijections on U are in $\mathcal{R}$, that $\mathcal{D} = \{\mathtt{dom}(R) | R \in \mathcal{R}\}$ is closed under complement, and that for all $R, S \in \mathcal{R}$, $D \in \mathcal{D}$ we have $\mathtt{id}_U, R^{-1}, R \circ S \in \mathcal{R}$ and $R \cap S, R\lceil D \in \mathcal{R}$. Let $\mathcal{U}$ be the structure with universe U, one constant denoting each element of U, and one predicate symbol denoting each $R \in \mathcal{R}$. Then:*

1. *The theory of $\mathcal{U}$ has quantifier-elimination.*
2. *Any set definable in $\mathcal{U}$ is in $\mathcal{D}$ and is $R(x, x)$ for some $R \in \mathcal{R}$.*

In order to give a flavor of our quantifier elimination procedure, we give an example: detailed proofs are in [1].

$$\exists x_4(R_1 x_1 x_4 \wedge R_2 x_2 x_4 \wedge \neg R x_3 x_4)$$

is equivalent to

$$\exists x_4(R_{1,4} x_1 x_4 \wedge R_{2,4} x_2 x_4 \wedge \neg R x_3 x_4)$$

where $D_4 = \mathtt{codom}(R_1) \cap \mathtt{codom}(R_2)$, $D_1 = R_1^{-1}(D_4)$, $D_2 = R_2^{-1}(D_4)$, and $R_{1,4} = R_1 \cap (D_1 \times D_4)$, $R_{2,4} = R_2 \cap (D_2 \times D_4)$. Note domain restriction here. It is equivalent to

$$\exists x_4(R_{1,4} x_1 x_4 \wedge R_{2,4} x_2 x_4 \wedge R_{1,2} x_1 x_2 \wedge \neg R x_3 x_4)$$

where $R_{1,2} = R_{2,4}^{-1} \circ R_{1,4}$. Note composition of relations here. It is equivalent to

$$\exists x_4(R_{1,4} x_1 x_4 \wedge R_{2,4} x_2 x_4 \wedge R_{1,2} x_1 x_2 \wedge \neg R' x_3 x_2)$$

where $R' = R_{2,4}^{-1} \circ R$. Note partial bijections here. It is equivalent to

$$\exists x_4(R_{1,4} x_1 x_4 \wedge R_{2,4} x_2 x_4) \wedge R_{1,2} x_1 x_2 \wedge \neg R' x_3 x_2$$

Using the properties of partial bijections, finally this is equivalent to

$$R_{1,2} x_1 x_2 \wedge \neg R' x_3 x_2.$$

In the proof we identify each constant symbol with the element $c \in U$ it denotes, and each predicate symbol with the relation $R \in \mathcal{R}$ it denotes. We prove quantifier elimination for $\mathcal{U}$ as defined in [8], Sects. 3.1 and 3.2, namely that each formula A with possibly free variables in the language $U \cup \mathcal{R}$ is equivalent in $\mathcal{U}$ to some formula B in the same language but without quantifiers. We will in fact prove a little more, namely that B may be chosen without constants.

We derive some closure properties for $\mathcal{R}$. For any $D \in \mathcal{D}$ we have $\mathtt{id}_D = (\mathtt{id}_{\mathcal{M}}\lceil D) \in \mathcal{R}$. We will abbreviate $\mathtt{id}_D(x, x)$ with $(x \in D)$. $\mathcal{D}$ is closed under intersection, because if $D, E \in \mathcal{D}$ then $\mathtt{id}_D, \mathtt{id}_E \in \mathcal{R}$, hence $\mathtt{id}_{D \cap E} = (\mathtt{id}_D \cap \mathtt{id}_E) \in \mathcal{R}$ and $D \cap E \in \mathcal{D}$. $\mathcal{D}$ includes $\mathcal{M} = \mathtt{dom}(\mathtt{id}_{\mathcal{M}})$ and it is closed under complement, therefore $\mathcal{D}$ is closed under all boolean operations. For all $x \in \mathcal{M}$ the partial bijection $\mathtt{id}_{\{x\}}$ is finite and by assumption it is in $\mathcal{R}$: thus, all singletons

of $\mathcal{M}$ are in $\mathcal{D}$. Assume $R \in \mathcal{R}$ and $D \in \mathcal{D}$: then $\mathtt{codom}(R) = \mathtt{dom}(R^{-1}) \in \mathcal{D}$ and $R(D) = (R\lceil D)(D) = \mathtt{codom}(R\lceil D) \in \mathcal{D}$. If $R \in \mathcal{R}$ and $D, E \in \mathcal{D}$, then $R \cap (D \times E) \in \mathcal{R}$ follows from $R \cap (D \times E) = (((R\lceil D)^{-1})\lceil E)^{-1}$.

In the next statement, recall that we defined the relation composition in the same order as function composition.

Lemma 7 (Composition and Product). *Let R, S be any relation and D, E, F be any sets. Assume $R(D) \cap S^{-1}(F) \subseteq E$. Then composition and Cartesian product commute: $(S \cap (E \times F)) \circ (R \cap (D \times E)) = (S \circ R) \cap (D \times F)$.*

6.1 A Notion of Normal Form for the Language $\mathcal{R}$

Let $n > 0$ be any positive integer. Assume $R \in \mathcal{R}$ and $i, j \in \{1, \ldots, n\}$. We call any formula $R(x_i, x_j)$ a *positive* $(\mathcal{R}, n)$-atom and any formula $\neg R(x_i, x_j)$ a *negative* $(\mathcal{R}, n)$-atom. A $(\mathcal{R}, n)$-atom is either a positive or a negative $(\mathcal{R}, n)$-atom. A $(\mathcal{R}, n)$-propositional formula is any formula obtained from positive $(\mathcal{R}, n)$-atoms by repeatedly applying disjunction and negation. Any $(\mathcal{R}, n)$-propositional formula has free variables in $x_0, \ldots, x_{n-1}$. We denote by $\mathcal{A}_n$ the set of $(\mathcal{R}, n)$-propositional formula, and by $\mathcal{H}_n$ the set of n-ary predicates definable in $\mathcal{U}$.

Our goal is to prove that for all $n \in \mathtt{Nat}$, any first-order predicate P of $\mathcal{R}$ is definable by some $A \in \mathcal{A}_n$, and if $n = 1$ then $P \in \mathcal{D}$. We have to prove that formulas of $\mathcal{A}_n$ are closed under existential.

This is the plan of the proof. We will define a notion of $(\mathcal{R}, n)$-normal form for formulas of $\mathcal{A}_n$, and prove that every $A \in \mathcal{A}_n$ has some $(\mathcal{R}, n)$-normal form. Then we will prove that if $A \in \mathcal{A}_n$ is in $(\mathcal{R}, n)$-normal form, then $\exists x_n.A$ (with possibly free variables) may be expressed in $\mathcal{A}_{n-1}$ in one of the following ways: either as some finite disjunction $A[c_1/x_1] \vee \ldots \vee A[c_k/x_n]$ for some constants $c_1, \ldots, c_k \in U$, or by the formula $B \in \mathcal{A}_{n-1}$, obtained from A by erasing all $(\mathcal{R}, n)$-atoms including x_n.

Assume $n > 0$ is any positive integer. Let $\mathcal{G}$ be any binary relation on $\{1, \ldots, n\}$. A $(\mathcal{G}, n)$-*family* is any family $\mathcal{F} = \{R_{i,j}(x_i, x_j) | (i, j) \in \mathcal{G}\}$ of *positive* $(\mathcal{R}, n)$-atoms such that $\mathtt{dom}(R_{i,j}) = \mathtt{dom}(R_{i,k})$ for all $i, j, k = 1, \ldots, n$. $\mathcal{F}$ is a *symmetric family* if $\mathcal{G}$ is a symmetric relation, and for all $(i, j) \in \mathcal{G}$ we have $R_{i,j} = R_{i,j}^{-1}$. $\mathcal{F}$ is a *equivalence family* if $\mathcal{G}$ is an equivalence relation, and for all $i = 1, \ldots, n$ we have $R_{i,i} = \mathtt{id}_{D_i}$ for some D_i, for all $(i, j) \in \mathcal{G}$ we have $R_{i,j} = R_{i,j}^{-1}$, for all $(i, j), (j, k) \in \mathcal{G}$ we have $R_{j,k} \circ R_{i,j} = R_{i,k}$. In this case $D_i = \mathtt{dom}(R_{i,j})$ for all $i, j = 1, \ldots, n$ and we call $D_1, \ldots, D_n$ the domains of the family.

A $(\mathcal{G}, n)$-*symmetric conjunction* is any conjunction of a $(\mathcal{G}, n)$-symmetric family. A $(\mathcal{G}, n)$-*equivalence conjunction* is any conjunction of a $(\mathcal{G}, n)$-equivalence family.

We recall some basic graph theory. We call an indirect, simple graph on $\{1, \ldots, n\}$ just a *graph*, and we represent it by any irreflexive and symmetric relation $\mathcal{G}$ on $\{1, \ldots, n\}$. A *simple cycle* in $\mathcal{G}$ is any sequence $\sigma = \{(i_0, i_1), (i_1, i_2), \ldots (i_{m-1}, i_m), (i_m, i_0)\} \subseteq \mathcal{G}$ of pairwise distinct $i_0, \ldots, i_m$ with $m \geq 2$. $\mathcal{G}$ is *acyclic* if $\mathcal{G}$ has no simple cycle. A path is any sequence

$\pi = \{(i_0, i_1), (i_1, i_2), \ldots (i_{m-1}, i_m)\}$ with pairwise distinct $i_1, \ldots, i_m$, with possibly $m = 0$. The connection relation on $\mathcal{G}$ is: "there is some path from i to j" In any acyclic graph $\mathcal{G}$ the path from i to j if it exists then it is unique. Given any equivalence relation $\mathcal{P}$, there is some minimal graph $\mathcal{G} \subseteq \mathcal{P}$ among those such that $\mathcal{P}$ is the smallest equivalence relation including $\mathcal{G}$. All these minimal graphs are acyclic.

Definition 6 ($(\mathcal{R}, n)$-Normal Forms). *$C = C_1 \wedge C_2$ is a $(\mathcal{R}, n)$-normal conjunction if C_1 is some conjunction of positive $(\mathcal{R}, n)$-atoms, C_2 is some conjunction of negative $(\mathcal{R}, n)$-atoms, and for some equivalence relation $\mathcal{P}$*

1. *C_1 is some $(\mathcal{P}, n)$-equivalence conjunction*
2. *for any $\neg S(x_i, x_j)$ in C_2 we have $i < j$*
3. *if $[n]_{\mathcal{P}} \neq \{n\}$ then x_n does not occur in C_2*

Any $A \in \mathcal{A}_n$ is an $(\mathcal{R}, n)$-normal form if A is some disjunction of $(\mathcal{R}, n)$-normal conjunctions.

We first prove that any $(\mathcal{G}, n)$-symmetric conjunction, with $\mathcal{G}$ some acyclic graph, is equivalent to some $(\mathcal{P}, n)$-equivalence conjunction, where $\mathcal{P}$ is the reflexive and transitive closure of $\mathcal{G}$.

Lemma 8 (Transitive Closure Lemma). *Let $n > 0$ be any positive integer. Assume $\mathcal{G}$ is any acyclic graph on $\{1, \ldots, n\}$ and $A = \wedge_{i,j \in \mathcal{G}} R_{i,j}(x_i, x_j)$ is any $(\mathcal{G}, n)$-symmetric conjunction. Let $\mathcal{P}$ be the reflexive, symmetric and transitive closure of $\mathcal{G}$. Then A is equivalent to some unique $(\mathcal{P}, n)$-equivalence conjunction B whose family of atoms extends the family of atoms of A.*

Now we prove that the $(\mathcal{P}, n)$-equivalence conjunctions are closed under conjunction with a positive $(\mathcal{R}, n)$-atom $R(x_i, x_j)$. For all $i = 1, \ldots, n$, we denote with $[i]_{\mathcal{P}}$ the equivalence class of i in $\mathcal{P}$.

Lemma 9 (Partition Lemma). *Assume $A = \wedge_{i,j \in \mathcal{P}} R_{i,j}(x_i, x_j)$ is any $(\mathcal{P}, n)$-equivalence conjunction, and $i, j \in \{1, \ldots, n\}$. Assume $D \in \mathcal{D}$ and $R \in \mathcal{R}$.*

1. *$A \wedge (x_i \in D)$ is equivalent to some $(\mathcal{P}, n)$-equivalence conjunction*
2. *Assume $[i]_{\mathcal{P}} = [j]_{\mathcal{P}}$. Then $A \wedge R(x_i, x_j)$ is equivalent to some $(\mathcal{P}, n)$-equivalence conjunction*
3. *Assume $[i]_{\mathcal{P}} \neq [j]_{\mathcal{P}}$ and $\mathtt{dom}(R) = \mathtt{dom}(R_{i,i})$ and and $\mathtt{codom}(R) = \mathtt{dom}(R_{j,j})$. Then $A \wedge R(x_i, x_j)$ is equivalent to some $(\mathcal{P}, n)$-equivalence conjunction*
4. *Any $A \wedge R(x_i, x_j)$ is equivalent to some $(\mathcal{P}, n)$-equivalence conjunction*

6.2 A Quantifier Elimination Result for $\mathcal{R}$

Now we prove a quantifier-elimination result for the language with symbols the binary predicates in $\mathcal{R}$, using Lemmas 6 and 9.

Lemma 10 (Quantifier Elimination for $\mathcal{R}$). *Let $n > 0$ be any positive integer.*

1. *Any finite conjunction of positive $(\mathcal{R}, n)$-atoms has some $(\mathcal{P}, n)$-equivalence form.*
2. *Any finite conjunction of positive and negative $(\mathcal{R}, n)$-atoms has some $(\mathcal{P}, n)$-equivalence form.*
3. *If A is some finite conjunction of positive and negative $(\mathcal{R}, n)$-atoms, then $\exists x_n.A$ is equivalent to some $B \in \mathcal{A}_{n-1}$.*
4. *If $A \in \mathcal{A}_n$, then $\exists x_n.A$ is equivalent to some $B \in \mathcal{A}_{n-1}$.*

We may now finish the proof of Theorem 2.

7 Main Theorem

Let $\mathcal{R}$, $\mathcal{D}$ as in Sect. 5. From the properties of the partial bijections in $\mathcal{R}$ and from the quantifier elimination result (Sect. 6) we derive our main result.

Theorem 3 (Counterexample to Brotherston-Simpson Conjecture). *Let H be the formula defined in Definition 1. Then H has a proof in $\mathtt{CLKID}^\omega(\Sigma_N, \Phi_N)$, and no proof in $\mathtt{LKID}(\Sigma_N, \Phi_N) + (0, s)$-axioms.*

Proof. The proof in $\mathtt{CLKID}^\omega$ is shown in Theorem 1. The non-provability in LKID is shown as follows. Any atomic formulas in $\mathcal{M}$ is in $\mathcal{R}$. By definition, $\mathcal{R}$ contains all finite bijections and is closed under restriction to any set $D \in \mathcal{D}$. Thus, by Lemma 6, $\mathcal{R}$ satisfies all hypothesis of Theorem 2. We deduce that all definable sets of $\mathcal{M}$ are in $\mathcal{D}$. By Lemma 3 point 4, all sets in $\mathcal{D}$ have a dyadic measure, and by Lemma 2 satisfy the induction schema. According to Definition 2.10 of [6], this is a sufficient condition for $\mathcal{M}$ being an Henkin model of $\mathtt{LKID}(\Sigma_N, \Phi_N)$. $\mathcal{M}$ satisfies the $(0, S)$-axioms by construction. $\mathcal{M}$ falsifies H by Lemma 1. □

8 Non-conservativity of Martin-Löf's Inductive Definition System

This section shows non-conservativity of LKID with respect to additional inductive predicates, by giving a counterexample.

We assume the inductive predicate $\leq$ and the production rules for it:

$$\frac{}{x \leq x} \qquad \frac{x \leq y}{x \leq sy}$$

We call the set of these production rules $\Phi_\leq$. Let 0-axiom be $\forall x \in N.\ sx \neq 0$. In $\mathtt{LKID}(\Sigma_N + \{\leq\}, \Phi_N + \Phi_\leq)$, we can show any number ≤ 0 is only 0.

Lemma 11. *0-axiom*, $Nx, Ny, x \leq y \vdash y = 0 \to x = 0$

The proof is in [1].

The next theorem shows 2-Hydra is provable in LKID with $\leq$.

Theorem 4. *0-axiom* $\vdash H$ *is provable in* $\mathtt{LKID}(\Sigma_N + \{\leq\}, \Phi_N + \Phi_{\leq})$.

We may show $\forall n.\ (n \geq x \wedge n \geq y \to p(x, y))$ by induction on n. The proof is given in [1] in case.

In the standard model, the truth of formula does not change when we extend the model with inductive predicates that do not appear in the formula. On the other hand, this is not the case for provability in Martin-Löf's inductive definition system LKID. Namely, a system may change the provability of a formula even when we add inductive predicates that do not appear in the formula. Namely, for a given system, the system with additional inductive predicates may not be conservative over the original system. Theorems 3 and 4 give such an example: the sequent 0-axiom $\vdash H$ is in the language of LKID but it is not provable in LKID, while it is provable in LKID extended with $\leq$.

9 Conclusion

We proved in Theorem 3 that $\mathtt{CLKID}^{\omega}$, the formal system of cyclic proofs ([6]) proves strictly more that LKID, Martin-Löf formal system of inductive definitions with classical logic. This settles an open question given in [6]. Our proof also shows that if we add more inductive predicates to LKID we may obtain a non-conservative extension (Theorem 4).

References

1. Berardi, S., Tatsuta, M.: The classic Martin-Löf's system of inductive definitions is not equivalent to cyclic proofs (Full Paper), unpublished draft (2017)
2. Brotherston, J.: Cyclic proofs for first-order logic with inductive definitions. In: Beckert, B. (ed.) TABLEAUX 2005. LNCS (LNAI), vol. 3702, pp. 78–92. Springer, Heidelberg (2005). doi:10.1007/11554554_8
3. Brotherston, J.: Sequent calculus proof systems for inductive definitions, Ph.D. thesis, Laboratory for Foundations of Computer Science School of Informatics University of Edinburgh (2006)
4. Brotherston, J., Bornat, R., Calcagno, C.: Cyclic proofs of program termination in separation logic. In: Proceedings of POPL 2008 (2008)
5. Brotherston, J., Distefano, D., Petersen, R.L.: Automated cyclic entailment proofs in separation logic. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE 2011. LNCS (LNAI), vol. 6803, pp. 131–146. Springer, Heidelberg (2011). doi:10.1007/978-3-642-22438-6_12
6. Brotherston, J., Simpson, A.: Sequent calculi for induction and infinite descent. J. Logic Comput. **21**(6), 1177–1216 (2011)

7. Brotherston, J., Gorogiannis, N., Petersen, R.L.: A generic cyclic theorem prover. In: Jhala, R., Igarashi, A. (eds.) APLAS 2012. LNCS, vol. 7705, pp. 350–367. Springer, Heidelberg (2012). doi:10.1007/978-3-642-35182-2_25
8. Enderton, H.B.: A Mathematical Introduction to Logic, 2nd edn. Academic Press, New York (2000)
9. Kirby, L., Paris, J.: Accessible independence results for Peano Arithmetic. Bull. Lond. Math. Soc. **14**, 285–293 (1982)
10. Martin-Löf, P.: Haupstatz for the intuitionistic theory of iterated inductive definitions. In: Proceedings of the Second Scandinavian Logic Symposium, pp. 179–216, North-Holland (1971)
11. Simpson, A.: Cyclic arithmetic is equivalent to Peano Arithmetic. In: Proceedings of Fossacs (2017)
12. Stratulat, S.: Structural vs. cyclic induction: a report on some experiments with Coq. In: SYNASC 2016: Proceedings of the 18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, pp. 27–34. IEEE Computer Society (2016)

Probability

On the Relationship Between Bisimulation and Trace Equivalence in an Approximate Probabilistic Context

Gaoang Bian[1,2(✉)] and Alessandro Abate[2(✉)]

[1] Google Inc., Mountain View, USA
gaoang@google.com
[2] Department of Computer Science, University of Oxford, Oxford, UK
aabate@cs.ox.ac.uk

Abstract. This work introduces a notion of approximate probabilistic trace equivalence for labelled Markov chains, and relates this new concept to the known notion of approximate probabilistic bisimulation. In particular this work shows that the latter notion induces a tight upper bound on the approximation between finite-horizon traces, as expressed by a total variation distance. As such, this work extends corresponding results for exact notions and analogous results for non-probabilistic models. This bound can be employed to relate the closeness in satisfaction probabilities over bounded linear-time properties, and allows for probabilistic model checking of concrete models via abstractions. The contribution focuses on both finite-state and uncountable-state labelled Markov chains, and claims two main applications: firstly, it allows an upper bound on the trace distance to be decided for finite state systems; secondly, it can be used to synthesise discrete approximations to continuous-state models with arbitrary precision.

1 Introduction

Often in formal verification one is interested in approximations of concrete models. Models are often built from experimental data that are themselves approximate, and taking approximations can reduce the size and complexity of the state space. Markov models in particular can be defined either syntactically as a transition structure (with states and matrices), or semantically as a random process whose trajectory satisfy the Markov property. Each representation gives rise to its own notions of approximation [1]: "the transition matrices have similar numbers and/or structure" vs "the trajectories have similar probability distributions", respectively. While the syntactic representation is used for computations and model checking with concrete numbers, often one is interested in results in terms of the semantics, e.g. "what is the probability of reaching a failure state within 100 steps". This gives practical value to studying how approximations in terms of transition matrices translate into approximations in terms of traces of the random process.

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 321–337, 2017.
DOI: 10.1007/978-3-662-54458-7_19

In this paper we build on the notion of ε-approximate probabilistic bisimulation, introduced in [13] as a natural extension to exact probabilistic bisimulation [12]. There, the notion of ε-approximate probabilistic bisimulation (or just ε-bisimulation) is defined in terms of the transition structure, and given ε the maximal ε-bisimulation relation can be computed for finite state systems with n states in $\mathcal{O}(n^7)$ time [13].

It is on the other hand of interest to explore what ε-bisimulation means in terms of trajectories. While ε-bisimulation does have characterizations (on countable state spaces) in terms of logics and games [12], this logic is branching in nature, and does not directly relate to the trajectory of the model as it leaks information about the state space (similarly to the difference between CTL and LTL), as illustrated in Fig. 1.

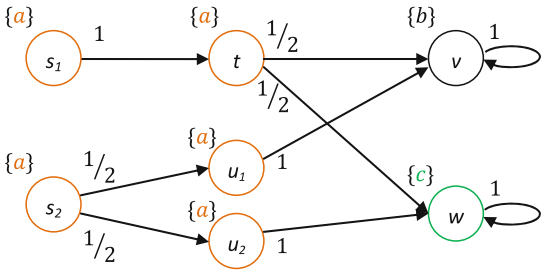


Fig. 1. Branching vs Linear Time Behaviour. In the Labelled Markov Chain below (cf. Sect. 2 for the LMC model), the states s_1, s_2 both emit traces $\langle\{a\},\{a\},\{b\}\rangle$ and $\langle\{a\},\{a\},\{c\}\rangle$ with probability 0.5 each, and hence s_1, s_2 have the same linear time behaviour. However, s_1, s_2 have different branching behaviour, since exclusively s_1 satisfies the PCTL formula $\mathrm{P}_{=1}[\,\mathrm{X}\,\mathrm{P}_{=0.5}[\,\mathrm{X}\,b\,]\,]$. Conversely, only s_2 satisfies $\mathrm{P}_{=0.5}[\,\mathrm{X}\,\mathrm{P}_{=1}[\,\mathrm{X}\,b\,]\,]$.

In this paper, we investigate what ε-approximate probabilistic bisimulation means in terms of trajectories. We will prove that for Labelled Markov Chains (over potentially uncountable state spaces), ε-bisimulation between two states places the tight upper bound of $1-(1-\varepsilon)^k$ (which is $\leq k\varepsilon$) on the total variation [18] between the distributions of length $k+1$ traces starting from those states, for all $k \in \mathbb{N}$. We will formulate these bounds by introducing the notion of $f(k)$-trace equivalence. As such, we extend the well known result that bisimulation implies trace equivalence in non-probabilistic systems to the context of approximate and probabilistic models (the exact probabilistic case having been considered in [7]).

One direct repercussion of our result is that it provides a method to efficiently bound the total variation of length k traces from two finite-state LMCs (or two states in an LMC), since the aforementioned result in [13] can be used to decide or to compute ε-bisimulation between two states in polynomial time. We will also apply our results to the quantitative verification of continuous-state Markov models [2,3,16], improving on the current class of properties approximated and the corresponding approximation errors.

Related Work. Literature on approximations of (finite-state) Markov models can be distinguished into two main branches: one focusing on one-step similarity, the other dealing with trace distances. One-step similarity can be studied via the notion of probabilistic bisimulation, introduced in the context of finite-state models by [20], and related to lumpability in [23]. [13] discusses a notion of approximate bisimulation, related to quasi-lumpability conditions in [6,22]. From the perspective of process algebra, [17] studies operators on probabilistic transition systems that preserve the approximate bisimulation distance. The work in [13,21] is seminal in introducing notions of (exact) probabilistic simulation, much extended in subsequent literature.

On the other hand, there are a few papers studying the total variation distance over traces. [9] presents an algorithm for approximating the total variation of infinite traces of labelled Markov systems and prove the problem of deciding whether it exceeds a given threshold to be NP-hard. [8] shows that the undiscounted bisimilarity pseudometric is a (non-tight) upper bound on the total variation of infinite traces (like ε-bisimulation, the bisimilarity pseudometric is defined on the syntax of the model and there are efficient algorithms for computing it [8,10,24]). The contribution in this paper, focusing on finite traces rather than infinite traces, is that the total variation of finite traces is much less conservative, and moreover allows manipulating models under specific error bounds on length k traces, as we will show in Sect. 5.

[14] studies notions based on the total variation of finite and infinite traces: employing a different notion of ε-bisimulation than ours, it proves error bounds on trace distances, which however depend on additional properties of the structure of the transition kernel (as shown in Sect. 7 the error bound on reachability probabilities could go to 1 in two steps, for any $\varepsilon > 0$). Finite abstractions of continuous-state Markov models can be synthesised by notions that are variations of the ε-bisimulation in this work [1,3,16]. Tangential to our work, [11] shows that the total variation of finite traces can be statically estimated via repeated observations. [26] investigates ways of compressing Hidden Markov Models by searching for a smaller model that minimises the total variation of length-k traces of the two models.

Structure of this article. In Sect. 2, we introduce the reference model (labelled Markov chains – LMC – over general state spaces) and provide a definition of ε-bisimulation for LMCs. In Sect. 3, we introduce the notion of approximate probabilistic trace equivalence (and the derived notion of probabilistic trace distance), and discuss how it relates to bounded linear time properties, and to the notion of distinguishability. In Sect. 4, we present the main result: we will derive a tight upper bound on the probabilistic trace distance between ε-bisimilar states. In Sect. 5, we show how these results can be used to approximately model check continuous state systems, and Sect. 6 discusses a case study. In Sect. 7, we discuss an alternative notion of approximate probabilistic bisimulation that appears in literature and show that it cannot be used to effectively bound probabilistic trace distance.

In this work we offer sketches of the proof of some theorems, and omit the proof of other results: the complete proofs, as well as the details on the implementation of the Case Study, can be found in the Appendix of [4].

2 Preliminaries

2.1 Labelled Markov Chains

We will work with discrete-time Labelled Markov Chains (LMCs) over general state spaces. Known definitions of countable- or finite-state LMCs represent special instances of the general models we introduce next.

Definition 1 (LMC syntax). *A Labelled Markov Chain (LMC) is a structure* $\mathscr{M} = (S, \Sigma, \tau, L)$ *where:*

- S *is a (potentially uncountable) set of states.*
- $\Sigma \subseteq \mathcal{P}(S)$ *is a* Σ*-algebra over* S *representing the set of measurable subsets of* S*.*
- $\tau : S \times \Sigma \to [0, 1]$ *is a transition kernel. That is, for all* $s \in S$*,* $\tau(s, \cdot)$ *is a probability measure on the measure space* (S, Σ)*, and for all* $A \in \Sigma$ *we require* $\tau(\cdot, A)$ *to be* Σ*-measurable.*
- $L : S \to \mathcal{O}$ *labels each state* $s \in S$ *with a subset of atomic propositions from* AP*, where* $\mathcal{O} = 2^{\mathrm{AP}}$*.* L *is required to be* Σ*-measurable, and we will assume* AP *to be finite.*

$L(s)$ captures all the observable information at state $s \in S$: this drives our notion relating pairs of states, and we characterise properties over the codomain of this function.

Definition 2 (LMC semantics). *Let* $\mathscr{M} = (S, \Sigma, \tau, L)$ *be a LMC. Given an initial distribution* p_0 *over* S*, the state of* $\mathscr{M}$ *at time* $t \in \mathbb{N}$ *is a random variable* $\mathscr{M}_t^{p_0}$ *over* S*, such that*

$$\mathbb{P}\left[\mathscr{M}_0^{p_0} \in A_0\right] = p_0(A_0),$$

$$\mathbb{P}\left[\mathscr{M}_0^{p_0} \in A_0, \cdots, \mathscr{M}_k^{p_0} \in A_k\right] = \int_{y_0 \in A_0} p_0(\mathrm{d}y_0) \cdot \int_{y_1 \in A_1} \tau(y_0, \mathrm{d}y_1) \cdots \int_{y_{k-1} \in A_{k-1}} \tau(y_{k-2}, \mathrm{d}y_{k-1}) \cdot \tau(y_{k-1}, A_k),$$

for all $k \in \mathbb{N}\backslash\{0\}, A_k \in \Sigma$*, where of course* $\tau(y_{k-1}, A_k) = \int_{y_k \in A_k} \tau(y_{k-1}, \mathrm{d}y_k)$*.*

Models in related work. A body of related literature works with labelled MDPs, which are more general models allowing a non-deterministic choice $u \in \mathcal{U}$ (for some finite $\mathcal{U}$) of the transition kernel τ_u at each step. This choice is made by a "policy" that probabilistically selects u based on past observations of the process. Whilst we will ignore non-determinism and work with LMCs for simplicity, our results can be adapted to labelled MDPs by quantifying over all

policies, or over all choices u for properties like ε-bisimulation, in order to remove the non-determinism. The seminal work on bisimulation and ε-bisimulation dealt with models known as LMPs [12]. LMPs allow for non-determinism (like labelled MDPs) but their states are unlabelled and at each step they have a probability of halting. For the study of bisimulation in this work, LMPs can be considered as a simplification of labelled MDPs to the case $\mathcal{O} = \{\emptyset, \{\text{halted}\}\}$.

2.2 Exact and Approximate Probabilistic Bisimulations

The notion of approximate probabilistic bisimulation (in this work just ε-bisimulation) is a structural notion of closeness, based on the stronger notion of exact probabilistic bisimulation [12]. We discuss both next. Considering a binary relation R over set X, we say that a subset $\tilde{S} \subseteq X$ is R-closed if $\tilde{S}$ contains its own image under R. That is, if $R(\tilde{S}) \coloneqq \{y \in X \mid x \in \tilde{S},\ x R\, y\} \subseteq \tilde{S}$.

Definition 3 (Exact probabilistic bisimulation). *Let $\mathscr{M} = (S, \Sigma, \tau, L)$ be a LMC. For $\varepsilon \in [0,1]$, an equivalence relation $R \subseteq S \times S$ over the state space is an exact probabilistic bisimulation relation if*

$$\forall (s_1, s_2) \in R, \quad \text{we have that } L(s_1) = L(s_2),$$
$$\forall (s_1, s_2) \in R,\ \forall \tilde{T} \in \Sigma \text{ s.t. } \tilde{T} \text{ is } R\text{-closed}, \quad \text{we have that } \tau(s_1, \tilde{T}) = \tau(s_2, \tilde{T}).$$

A pair of states $s_1, s_2 \in S$ are said to be (exactly probabilistically) bisimilar if there exists an exact probabilistic bisimulation relation R such that $s_1 R\, s_2$.

Note that since R is an equivalence relation, R-closed sets are exactly the unions of whole equivalence classes.

Next, we adapt the notion of ε-bisimulation (as discussed in [13] for LMPs over countable state spaces) to LMCs over general spaces.

Definition 4 (ε-bisimulation). *Let $\mathscr{M} = (S, \Sigma, \tau, L)$ be a LMC. For $\varepsilon \in [0,1]$, a symmetric binary relation $R_\varepsilon \subseteq S \times S$ over the state space is an ε-approximate probabilistic bisimulation relation (or just ε-bisimulation relation) if*

$$\forall T \in \Sigma, \quad \text{we have } R_\varepsilon(T) \in \Sigma, \tag{1}$$
$$\forall (s_1, s_2) \in R_\varepsilon, \quad \text{we have } L(s_1) = L(s_2), \tag{2}$$
$$\forall (s_1, s_2) \in R_\varepsilon,\ \forall T \in \Sigma, \quad \text{we have } \tau(s_2, R_\varepsilon(T)) \geq \tau(s_1, T) - \varepsilon. \tag{3}$$

Two states $s_1, s_2 \in S$ are said to be ε-bisimilar if there exists an ε-bisimulation relation R_ε such that $s_1 R_\varepsilon s_2$.

The condition raised in (3) could be understood intuitively as "for any move that s_1 can take (say, into set T), s_2 can match it with higher likelihood over the corresponding set $R_\varepsilon(T)$, up to ε tolerance." Notice that (1) is not a necessary requirement for countable state models, but for uncountable state models it is needed to ensure that $R_\varepsilon(T)$ is measurable and $\tau(s_2, R_\varepsilon(T))$ is defined in (3).

[13] showed that in countable state spaces, 0-approximate probabilistic bisimulation corresponds to exact probabilistic bisimulation. On uncountable state spaces, not every exact probabilistic bisimulation relation is a 0-bisimulation relation because of the additional measurably requirement, but we still have that 0-bisimulation implies exact probabilistic bisimulation.

Theorem 1. *Let $\mathscr{M} = (S, \Sigma, \tau, L)$ be a LMC, and let $s_1, s_2 \in S$. If s_1, s_2 are 0-bisimilar, then they are (exactly, probabilistically) bisimilar.*

Although above s_1, s_2 are required to belong to the state space of a given LMC, the notions of exact- and ε-bisimulation can be extended to hold over pairs of LMCs by combining their state spaces, as follows.

Definition 5 (ε-bisimulation of pairs of LMCs). *Consider two LMCs $\mathscr{M}_1 = (S_1, \Sigma_1, \tau_1, L_1)$ and $\mathscr{M}_2 = (S_2, \Sigma_2, \tau_2, L_2)$. Without loss of generality, assume that their state spaces S_1, S_2 are disjoint. The direct sum $\mathscr{M}_1 \oplus \mathscr{M}_2$ of $\mathscr{M}_1$ and $\mathscr{M}_2$ is the LMC formed by combining the state spaces of $\mathscr{M}_1$ and $\mathscr{M}_2$. Formally, $\mathscr{M}_1 \oplus \mathscr{M}_2 = (S_1 \uplus S_2, \sigma(\Sigma_1 \times \Sigma_2), \tau_1 \oplus \tau_2, L_1 \uplus L_2)$, where:*

- *$S_1 \uplus S_2$ is the union of S_1 and S_2 where we have assumed wlog (by relabelling if necessary) that S_1, S_2 are disjoint;*
- *$\sigma(\Sigma_1 \times \Sigma_2)$ is the smallest σ-algebra containing $\Sigma_1 \times \Sigma_2$;*
- $\tau_1 \oplus \tau_2(s, T) := \begin{cases} \tau_1(s,\ T \cap S_1) & \text{if } s \in S_1 \\ \tau_2(s,\ T \cap S_2) & \text{if } s \in S_2 \end{cases} for s \in S_1 \uplus S_2, T \in \sigma(\Sigma_1 \times \Sigma_2)$;
- $L_1 \uplus L_2(s) := \begin{cases} L_1(s) & \text{if } s \in S_1 \\ L_2(s) & \text{if } s \in S_2 \end{cases} for s \in S_1 \uplus S_2.$

Let $s_1 \in S_1$, $s_2 \in S_2$. We say that s_1, s_2 are ε-bisimilar iff s_1, s_2 are ε-bisimilar as states in the direct sum LMC $\mathscr{M}_1 \oplus \mathscr{M}_2$.

Other Notions of ε-Bisimulation in Literature. There is an alternative, more direct, extension of exact probabilistic bisimulation in literature [1,3,14], which simply requires $|\tau(s_1, \tilde{T}) - \tau(s_2, \tilde{T})| \leq \varepsilon$ instead of the conditions in Definition 4. However, this requirement alone is too weak to guarantee properties that we later discuss (cf. Sect. 7).

3 Approximate Probabilistic Trace Equivalence for LMCs

In this section we introduce the concept of approximate probabilistic trace equivalence (or just $f(k)$-trace equivalence) to represent closeness of observable linear time behaviour. Based on the likelihood over traces of a given LMC, this notion depends on its operational semantics (cf. Definition 2), rather than on the structure of its transition kernel (as in the case of approximate bisimulation). The notion can alternatively be thought of inducing a distance among traces, as elaborated below.

Definition 6 (Trace likelihood). *Let $\mathscr{M} = (S, \Sigma, \tau, L)$ be an LMC, $s_0 \in S$, and $k \in \mathbb{N}$. Let* TRACE *denote a set of traces (each of length $k+1$), taking values in time over 2^{AP}, so that* $\mathrm{TRACE} \subseteq \mathcal{O}^{k+1}$. *Denote with $P_k(s_0, \mathrm{TRACE})$ the probability that the LMC $\mathscr{M}$, given an initial state s_0, generates any of the runs $\langle \alpha_0, \cdots, \alpha_k \rangle \in$ TRACE, namely*

$$P_k(s_0, \mathrm{TRACE}_k) = \sum_{\substack{\langle \alpha_0, \cdots, \alpha_k \rangle \\ \in \mathrm{TRACE}_k}} \mathbb{P}\big[\mathscr{M}_0^{s_0} \in L^{-1}(\{\alpha_0\}), \cdots, \mathscr{M}_k^{s_0} \in L^{-1}(\{\alpha_k\})\big],$$

where $\mathscr{M}_t^{s_0}$ is the state of $\mathscr{M}$ at step t, with a degenerate initial distribution p_0 that is concentrated on point s_0 (cf. Definition 2).

As intuitive, we consider traces of length $k+1$ (rather than of length k) because a length $k+1$ trace is produced by one initial state and precisely k transitions. Notice that the set of sequences of states generating TRACE is measurable, being defined via a measurable map L over a finite set of traces.

Definition 7 (Total variation [15]). *Let $(Z, \mathcal{G})$ be a measure space where $\mathcal{G}$ is a σ-algebra over Z, and let μ_1, μ_2 be probability measures over $(Z, \mathcal{G})$. The total variation between μ_1, μ_2 is $d_{\mathrm{TV}}(\mu_1, \mu_2) \coloneqq \sup_{A \in \mathcal{G}} |\mu_1(A) - \mu_2(A)|$.*

Definition 8 ($f(k)$-trace equivalence). *Let $\mathscr{M} = (S, \Sigma, \tau, L)$ be a LMC. For a non-decreasing function $f : \mathbb{N} \to [0, 1]$, we say that states $s_1, s_2 \in S$ are $f(k)$-approximate probabilistic trace equivalent if for all $k \in \mathbb{N}$,*

$$d_{\mathrm{TV}}\big(P_k(s_1, \cdot)\,, P_k(s_2, \cdot)\,\big) \leq f(k),$$

or alternatively if over $\mathrm{TRACE} \subseteq \mathcal{O}^{k+1}$,

$$|P_k(s_1, \mathrm{TRACE}) - P_k(s_2, \mathrm{TRACE})| \leq f(k).$$

The condition on monotonicity follows from the requirement on the total variation distance, which is defined over a product output space and necessarily accumulates over time. The notion of $f(k)$-trace equivalence can be used to relate states from two different LMCs, much in the same way as ε-bisimulation.

One can introduce the notion of *probabilistic trace distance* between pairs of states s_1, s_2 as

$$\min\{f(k) \geq 0 \mid s_1 \text{ is } f(k)\text{-trace equivalent to } s_2\} = d_{\mathrm{TV}}\big(P_k(s_1, \cdot)\,, P_k(s_2, \cdot)\,\big).$$

Notice that the RHS is clearly a pseudometric. We discuss the development of tight bounds on the probabilistic trace distance in Sect. 4.

3.1 Interpretation and Application of ε-Trace Equivalence

The notion of ε-trace equivalence subsumes closeness of finite-time traces, and can be interpreted in two different ways. Firstly, ε-trace equivalence leads to closeness of satisfaction probabilities over bounded-horizon linear time properties, e.g. bounded LTL formulae, as follows.

Theorem 2. *Let $\mathscr{M} = (S, \Sigma, \tau, L)$ be an LMC, and let $s_1, s_2 \in S$ be ε-trace equivalent. Let ψ be any bounded LTL property over a k-step time horizon, defined within the LTL fragment $\phi = \phi_1 \mathrm{U}^{\leq t} \phi_2 \mid a \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \neg\phi_1$ for $t \leq k$. Then,*

$$\Big| \mathbb{P}[s_1 \models \psi] - \mathbb{P}[s_2 \models \psi] \Big| \leq f(k),$$

where $P[s \models \psi]$ is the probability that starting from state s, the LMC satisfies property ψ.

Proof. Formula ψ is satisfied by a specific set of length $k+1$ traces. □

Alternatively, via its connection to the notion of total variation, ε-trace *distance* leads to the notion of distinguishability of the underlying LMC, namely the ability (of an agent external to the model) to distinguish a model by observing its traces.

Theorem 3. *Let s_1, s_2 be two states of an LMC. Suppose one of them is selected by a secret fair coin toss. An external agent guesses which one has been selected by observing a trace of length $k+1$ emitted from the unknown state. Then, an optimal agent guesses correctly with probability*

$$\frac{1}{2} + \frac{1}{2} f(k),$$

with $f(k) = d_{\mathrm{TV}}\big(P_k(s_1, \cdot), P_k(s_2, \cdot)\big)$ being the probabilistic trace distance.

4 ε-Probabilistic Bisimulation Induces Approximate Probabilistic Trace Equivalence

In this section we present the main result: we show that ε-bisimulation induces a tight upper bound on the probabilistic trace distance, quantifiable as $(1 - (1-\varepsilon)^k)$. This translates to a guarantee on all the properties implied by ε-trace equivalence, such as closeness of satisfaction probabilities for bounded linear time properties. In addition, since for finite state LMPs the maximal ε-bisimulation relation can be computed in $\mathcal{O}(|S|^7)$ time [13], this result allows to establish an upper bound on the probabilistic trace distance with the same time complexity.

Theorem 4 (ε-bisimulation implies $(1 - (1-\varepsilon)^k)$-trace equivalence). *Let $\mathscr{M} = (S, \Sigma, \tau, L)$ be a LMC. If $s_1, s_2 \in S$ are ε-bisimilar, then s_1, s_2 are $(1 - (1-\varepsilon)^k)$-trace equivalent.*

Proof (Sketch). The full proof, developed for LMCs over uncountable state spaces, can be found in Appendix C of [4]. Here we offer a sketch of proof, employing the finite-state LMP in Fig. 2 as an illustrating example (where for simplicity we have omitted the labels for internal states, which can as well be labelled with $\emptyset \in \mathcal{O}$).

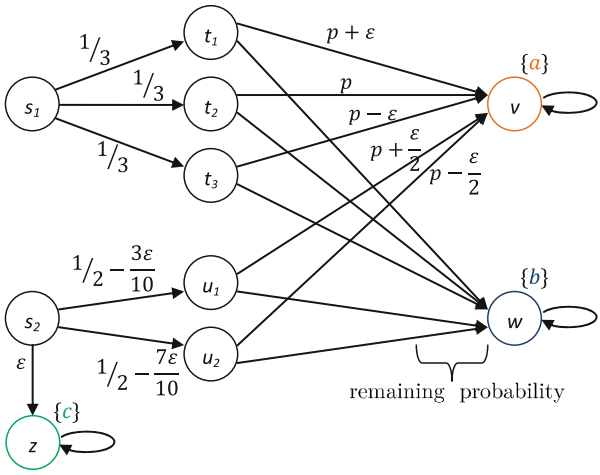


Fig. 2. LMC for the proof of Theorem 4.

The maximal (i.e. coarsest) ε-bisimulation relation R_ε is obtained by pairs of states within the sets

$$\{t_1, t_2, u_1\}, \{t_2, t_3, u_2\}, \{s_1, s_2\}, \{v\}, \{w\}, \{z\}.$$

We would like to prove that these ε-bisimilar states are also ε-trace equivalent. In the full proof, we will show this by induction on the length of the trace, for all ε-bisimilar states at the same time. In this sketch proof, we aim to illustrate the induction step by showing how to bound

$$\left|P_3\left(s_1, \Diamond^{\leq 3} a\right) - P_3\left(s_2, \Diamond^{\leq 3} a\right)\right|,$$

where $\Diamond^{\leq k} a$ is the set of traces of length $k+1$, which reach a state labelled with a (which in this case is just state v). The idea is to match each of the outgoing transitions from s_1 to an outgoing transition from s_2 and to an ε-bisimilar state. Specifically, we explicitly write

$$\begin{aligned} P_3\left(s_1, \Diamond^{\leq 3} a\right) = \frac{1}{3} P_2\left(t_1, \Diamond^{\leq 2} a\right) + \frac{1}{6} P_2\left(t_2, \Diamond^{\leq 2} a\right) \\ + \frac{1}{6} P_2\left(t_2, \Diamond^{\leq 2} a\right) + \frac{1}{3} P_2\left(t_3, \Diamond^{\leq 2} a\right), \end{aligned} \tag{4}$$

and respectively

$$\begin{aligned} P_3\left(s_2, \Diamond^{\leq 3} a\right) = \left(\frac{1}{3} - \frac{3\varepsilon}{10}\right) \cdot P_2\left(u_1, \Diamond^{\leq 2} a\right) + \frac{1}{6} P_2\left(u_1, \Diamond^{\leq 2} a\right) \\ + \frac{1}{6} P_2\left(u_2, \Diamond^{\leq 2} a\right) + \left(\frac{1}{3} - \frac{7\varepsilon}{10}\right) \cdot P_2\left(u_2, \Diamond^{\leq 2} a\right). \end{aligned} \tag{5}$$

We then match-off the terms in the expansions for $P_3\left(s_1, \Diamond^{\leq 3} a\right)$ and $P_3\left(s_2, \Diamond^{\leq 3} a\right)$, one term at a time. We use the induction hypothesis to argue

that the probabilities in the matched terms are $(1-(1-\varepsilon)^k)$*-close to each other (here* $k=1$*), since they concern* ε*-bisimilar states. That is,*

$$\left|P_2\left(t_1, \Diamond^{\leq 2} a\right) - P_2\left(u_1, \Diamond^{\leq 2} a\right)\right| \leq 1-(1-\varepsilon)^k$$
$$\left|P_2\left(t_2, \Diamond^{\leq 2} a\right) - P_2\left(u_1, \Diamond^{\leq 2} a\right)\right| \leq 1-(1-\varepsilon)^k$$
$$\ldots$$

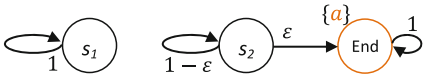
The total amount of difference between the matching coefficients is no more than ε*. It can be shown (Lemma 1 in Appendix C of [4]) that these conditions guarantee the required bound on* $\left|P_3\left(s_1, \Diamond^{\leq 3} a\right) - P_3\left(s_2, \Diamond^{\leq 3} a\right)\right|$*.*

The main difficulty is choosing a suitable decomposition of $P_3\left(s_1, \Diamond^{\leq 3} a\right)$ *and* $P_3\left(s_2, \Diamond^{\leq 3} a\right)$*. This is non-trivial since in (4), the* $1/3$ *probability of transitioning into* t_2 *had to be broken up into two terms with* $1/6$ *probability each. However, we can tackle this issue using an extension of Hall's Matching Theorem [5] (cf. Appendix C of [4]). It is then relatively straight forward to adapt this proof to LMCs on uncountable state spaces by converging on integrals with simple functions.* □

We now show that the expression for the induced bound on probabilistic trace distance proved in Theorem 4, namely $(1-(1-\varepsilon)^k)$, is tight, in the sense that for any k and ε, the bound can be attained by some pair of ε-bisimilar states in some LMC. In other words, it is not possible to provide bounds on the induced approximation level for traces, that are smaller than the expression discussed above and that are valid in general.

Theorem 5. *For any* $\varepsilon \geq 0$*, there exists a LMC* $\mathscr{M} = (S, \Sigma, \tau, L)$ *and states* $s_1, s_2 \in S$ *such that* s_1, s_2 *are* ε*-bisimilar, and for all* $k \in \mathbb{N}$ *there exists a set* TRACE *of length* $k+1$ *traces s.t.* $|P_k(s_1, \text{TRACE}) - P_k(s_2, \text{TRACE})| = 1-(1-\varepsilon)^k$*.*

Proof. Select $\varepsilon \geq 0$ *and consider the following LMC:*



Here s_1, s_2 *are* ε*-bisimilar, and for all* $k \in \mathbb{N}$*,* $P_k\left(s_1, \Diamond^{\leq k} a\right) = 0$*, whereas* $P_k\left(s_2, \Diamond^{\leq k} a\right) = \sum_{i=1}^{k} (1-\varepsilon)^{i-1} \varepsilon = 1-(1-\varepsilon)^k$*.* □

The result in Theorem 4 can be viewed as an extension of the known fact that bisimulation implies trace equivalence in non-probabilistic transition systems. Similar to the deterministic case, the converse of Theorem 4 does not hold.

Theorem 6. $(1-(1-\varepsilon)^k)$*-trace equivalence does not imply* ε*-bisimulation.*

Proof. In Fig. 1, states s_1, s_2 *are not* ε*-bisimilar for any* $\varepsilon < 1/2$*, yet their probabilistic trace distance is equal to* 0*.* □

This example shows that ε-bisimulation cannot be used to effectively estimate the probabilistic trace distance between individual states. In particular, while the $(1-(1-\varepsilon)^k)$-bound on probabilistic trace distance discussed above is tight as a uniform bound, it is not tight for individual pairs of states.

5 Application to Model Checking of Continuous-State LMCs

Suppose we are given an LMC $\mathscr{M}^{\mathcal{C}} = \left(S^{\mathcal{C}}, \Sigma^{\mathcal{C}}, \tau^{\mathcal{C}}, L^{\mathcal{C}}\right)$, which we shall refer to as the "concrete" model, possibly over a continuous state space. We are interested in calculating its probability of satisfying a given LTL formula, starting from certain initial states. One approach is to construct a finite-state LMC $\mathscr{M}^{\mathcal{A}} = \left(S^{\mathcal{A}}, \Sigma^{\mathcal{A}}, \tau^{\mathcal{A}}, L^{\mathcal{A}}\right)$ (the "abstract model") that can be related to $\mathscr{M}^{\mathcal{C}}$ (in a way to be made precise shortly). Probabilistic model checking can then be run over $\mathscr{M}^{\mathcal{A}}$ using standard tools for finite-state models such as PRISM [19], and since $\mathscr{M}^{\mathcal{A}}$ is related to $\mathscr{M}^{\mathcal{C}}$, this leads to approximate outcomes that are valid for $\mathscr{M}^{\mathcal{C}}$. The above approach has been studied in several papers [2,3,16], and the method for constructing $\mathscr{M}^{\mathcal{A}}$ from $\mathscr{M}^{\mathcal{C}}$ is to raise smoothness assumptions on the kernel $\tau^{\mathcal{C}}$ of $\mathscr{M}^{\mathcal{C}}$, and to partition the state space $S^{\mathcal{C}}$, thus obtaining $S^{\mathcal{A}}$ and $\tau^{\mathcal{A}}$ (the sigma algebra and labels being directly inherited).

In this section we will demonstrate the application of our results. We will employ ε-bisimulation to relate $\mathscr{M}^{\mathcal{A}}$ and $\mathscr{M}^{\mathcal{C}}$, and use our results to bound their trace distance. This method produces tighter error bounds, for a broader class of properties, than are currently established in literature. The first step is to establish simpler conditions that guarantee ε-bisimulation between $\mathscr{M}^{\mathcal{C}}$ and $\mathscr{M}^{\mathcal{A}}$.

Theorem 7. *Let $\varepsilon \in [0,1]$, and suppose there exists a finite measurable partition $\mathcal{Q}_\varepsilon = \{P_1, \cdots, P_N\}$ of $S^{\mathcal{C}}$ such that for all $P \in \mathcal{Q}_\varepsilon$, $s_1, s_2 \in P$, we have that $L^{\mathcal{C}}(s_1) = L^{\mathcal{C}}(s_2)$ and*[1]

$$\max_{J \subseteq \{1,\cdots,N\}} \left| \tau^{\mathcal{C}}\left(s_1, \bigcup_{j \in J} P_j\right) - \tau^{\mathcal{C}}\left(s_2, \bigcup_{j \in J} P_j\right) \right| \leq \varepsilon.$$

Assume wlog $P_i \neq \emptyset$, and for each $i \in \{1, \cdots, N\}$ choose a representative point $s_i^{\mathcal{C}} \in P_i$. Consider the abstract model to be $\mathscr{M}^{\mathcal{A}} = \left(S^{\mathcal{A}}, \Sigma^{\mathcal{A}}, \tau^{\mathcal{A}}, L^{\mathcal{A}}\right)$ formed by merging each P_i into $s_i^{\mathcal{C}}$. Formally,

- $S^{\mathcal{A}} = \{s_1^{\mathcal{A}}, \cdots, s_N^{\mathcal{A}}\}$.
- $\Sigma^{\mathcal{A}} = \mathcal{P}\left(S^{\mathcal{A}}\right)$.
- *$\tau^{\mathcal{A}}$ is such that $\tau^{\mathcal{A}}(s_i^{\mathcal{A}}, \{s_j^{\mathcal{A}}\}) = \tau^{\mathcal{C}}(s_i^{\mathcal{C}}, P_j)$.*
- $L^{\mathcal{A}}(s_i^{\mathcal{A}}) = L^{\mathcal{C}}(s_i^{\mathcal{C}})$.

Then, for all $i \in \{1, \cdots, N\}$, $s^{\mathcal{C}} \in P_i$, we have that $s^{\mathcal{C}}$ is ε-bisimilar to $s_i^{\mathcal{A}}$, and hence $1 - (1-\varepsilon)^k$-trace equivalent.

In practical terms the partition $\mathcal{Q}_\varepsilon$ can be straightforwardly constructed in many cases. As shown in Theorem 8, the approach in [2,3,25] generates a partition of $S^{\mathcal{C}}$ satisfying the conditions of Theorem 7. Thus, ε-bisimulation can be seen as the underlying reason for the closeness of probabilities of events.

[1] The left hand side is just $d_{\mathrm{TV}}(\mu_1, \mu_2)$ where for $i = 1, 2$, for $A \subseteq \mathcal{Q}_\varepsilon$, $\mu_i(A) := \tau^{\mathcal{C}}\left(s_i, \bigcup A\right)$.

Theorem 8. *Consider an LMC* $\mathscr{M}^{\mathcal{C}} = \left(S^{\mathcal{C}}, \Sigma^{\mathcal{C}}, \tau^{\mathcal{C}}, L^{\mathcal{C}}\right)$ *where* $S^{\mathcal{C}}$ *is a Borel subset of* $\mathbb{R}^d$*. Suppose that* $\tau^{\mathcal{C}}(s,T)$ *is of the form* $\int_{t\in T} f(s,t)\mathrm{d}t$*, so that for each state* $s \in S^{\mathcal{C}}$*,* $f(s,\cdot) : S^{\mathcal{C}} \to \mathbb{R}_0^+$ *is the probability density of the next state. Suppose further that* $f(\cdot,t)$ *is uniformly* K*-Lipschitz continuous for all* $t \in S^{\mathcal{C}}$*. That is, for some* $K \in \mathbb{R}$*, for all* $s_1, s_2, t \in S^{\mathcal{C}}$*,*

$$|f(s_1,t) - f(s_2,t)| \leq K \cdot \|s_1 - s_2\|.$$

For $A \in \Sigma^{\mathcal{C}}$ *(so* $A \subseteq \mathbb{R}^d$*), let* $\lambda(A)$ *be the volume of* A *and* $\delta(A) := \sup_{x_1,x_2\in A}\{\|x_1 - x_2\|\}$ *be the diameter of* A*. For any* $\varepsilon \in [0,1]$*, finite* $\lambda(S^{\mathcal{C}})$*, suppose partition* $\mathcal{Q} = \{P_1, \cdots, P_N\}$ *of* $S^{\mathcal{C}}$ *is such that*

$$\max_{j\in\{1,\cdots,N\}} \delta(P_j) \leq \frac{2\varepsilon}{K\lambda(S^{\mathcal{C}})}.$$

Then, we have that $\mathcal{Q}$ *satisfies the conditions of Theorem 7 and can be used to construct the abstract model* $\mathscr{M}^{\mathcal{A}}$*.*

There are a number of adaptations that could be made to this result. [16] improves a related approach by varying the size of each partition in response to the local Lipschitz constant, rather than enforcing a globally uniform K. Similarly to this paper, [3] also discusses the relation of approximate probabilistic bisimulation to the problem of generating the abstract model, but a strictly weaker definition of approximate probabilistic bisimulation is employed (cf. Sect. 7). Finally, note that using algorithms in [13], we can compute ε-bisimulation relations on $\mathscr{M}^{\mathcal{A}}$: this allows $\mathscr{M}^{\mathcal{A}}$ to be further compressed (at the cost of an additional ε_2 approximation), by merging the states that are ε_2-bisimilar to each other.

6 Case Study

Concrete Model. Consider the concrete model $\mathscr{M}^{\mathcal{C}} = \left(S^{\mathcal{C}}, \Sigma^{\mathcal{C}}, \tau^{\mathcal{C}}, L^{\mathcal{C}}\right)$, describing the weather forecast for a resort. Here $S^{\mathcal{C}} = \{0,1\} \times [0,1)$, $\Sigma^{\mathcal{C}} = \mathcal{B}(S^{\mathcal{C}})$, and the state at time t is (R_t, H_t), where

- $R_t \in \{0,1\}$ is a random variable representing whether it rains on day t,
- $H_t \in [0,1)$ is a random variable representing the humidity after day t.

Raining on day t causes it to become more likely to rain on day $t+1$, but it also tends to reduce the humidity, which causes it to become gradually less likely to rain in the future. The meteorological variations are encompassed by $\tau^{\mathcal{C}}$, which is such that the model evolves according to

$$\begin{aligned}
&\mathbb{P}(R_{t+1} \mid R_0, \cdots, R_t, H_0, \cdots, H_t) = \mathbb{P}(R_{t+1} \mid R_t, H_t) \\
&\qquad \sim \begin{cases} \mathrm{B}(\frac{1}{4} + \frac{3}{4}H_t) & \text{if } R_t = 1 \\ \mathrm{B}(\frac{3}{4}H_t) & \text{if } R_t = 0 \end{cases}, \\
&\mathbb{P}(H_{t+1} \mid R_0, \cdots, R_t, H_0, \cdots, H_t, R_{t+1}) = \mathbb{P}(H_{t+1} \mid H_t, R_{t+1}) \\
&\qquad \sim \begin{cases} \mathrm{U}\left[0, \frac{1+H_t}{2}\right) & \text{if } R_{t+1} = 1 \\ \mathrm{U}\left[\frac{H_t}{2}, 1\right) & \text{if } R_{t+1} = 0 \end{cases},
\end{aligned}$$

where $\mathrm{B}(p)$ is the Bernoulli distribution with probability p of producing 1, and $\mathrm{U}[a,b)$ is the uniform distribution over the real interval $[a,b)$. Finally, the states of the model are labelled according to whether it rains on that day, namely

$$L^{\mathcal{C}}((r,h)) = \begin{cases} \{\text{RAIN}\} & \text{if } r = 1 \\ \emptyset & \text{if } r = 0. \end{cases}$$

Given $\mathscr{M}^{\mathcal{C}}$ we are interested in computing the likelihood of events expressing meteorological predictions, given knowledge of present weather conditions.

Synthesis of the Abstract Model. Notice that $\mathscr{M}^{\mathcal{C}}$ does not directly satisfy the smoothness assumptions of Theorem 8, in view of the discrete/continuous structure of its state space and the discontinuous probability density resulting from the uniform distribution. Nonetheless, we can still construct $\mathscr{M}^{\mathcal{A}}$ by taking a sensible partition of $S^{\mathcal{C}}$ and proving that it satisfies the conditions of Theorem 7. Let $\mathcal{Q}_\varepsilon := \left\{P_{r,h} \;\middle|\; r \in \{0,1\},\ h \in \{0,\cdots,N-1\}\right\}$, where $P_{r,h} = \{r\} \times [{}^{h}/_{N}, {}^{h+1}/_{N})$.

Theorem 9. *For any $\varepsilon \in [0,1]$, by taking $N \geq {}^{2}/_{\varepsilon}$, we have that $\mathcal{Q}_\varepsilon$ satisfies the conditions of Theorem 7.*

Therefore, we may construct the abstract model using Theorem 7. We choose the abstract state $(r^{\mathcal{A}}, h^{\mathcal{A}}) \in S^{\mathcal{A}} := \{0,1\} \times \{0,\cdots,N-1\}$ to correspond to the partition $P_{r^{\mathcal{A}},h^{\mathcal{A}}} \in \mathcal{Q}_\varepsilon$, and within each partition we select the concrete state with the lowest H_t-coordinate to be the representative state. This produces the abstract model $\mathscr{M}^{\mathcal{A}} = \left(S^{\mathcal{A}}, \Sigma^{\mathcal{A}}, \tau^{\mathcal{A}}, L^{\mathcal{A}}\right)$, where $\Sigma^{\mathcal{A}} = \mathcal{P}(S^{\mathcal{A}})$, $L^{\mathcal{A}}((r,h)) = L^{\mathcal{C}}((r,h))$, and

$$\tau^{\mathcal{A}}\left((h_0,r_0),\{(h_1,r_1)\}\right) = \begin{cases} p_{\mathrm{R}_1|\mathrm{R}_0}(h_0)\cdot p_{H_1|\mathrm{R}_1}(h_0,h_1) & \text{if } r_0 = 1, r_1 = 1 \\ \left(1 - p_{\mathrm{R}_1|\mathrm{R}_0}(h_0)\right)\cdot p_{H_1|\neg\mathrm{R}_1}(h_0,h_1) & \text{if } r_0 = 1, r_1 = 0 \\ p_{\mathrm{R}_1|\neg\mathrm{R}_0}(h_0)\cdot p_{H_1|\mathrm{R}_1}(h_0,h_1) & \text{if } r_0 = 0, r_1 = 1 \\ \left(1 - p_{\mathrm{R}_1|\neg\mathrm{R}_0}(h_0)\right)\cdot p_{H_1|\neg\mathrm{R}_1}(h_0,h_1) & \text{if } r_0 = 0, r_1 = 0, \end{cases}$$

where

- $p_{\mathrm{R}_1|\mathrm{R}_0}(h_0) = \frac{1}{4} + \frac{3}{4}\frac{h_0}{N}$, and $p_{\mathrm{R}_1|\neg\mathrm{R}_0}(h_0) = \frac{3}{4}\frac{h_0}{N}$,
- $p_{H|\neg\mathrm{R}}(h_0,h_1) = \frac{2}{2N-h_0}\cdot\max\left(\min\left(h_1 + 1 - {}^{h_0}/_{2}, 1\right), 0\right)$,
- $p_{H|\mathrm{R}}(h_0,h_1) = \frac{2}{N+h_0}\cdot\max\left(\min\left(\frac{N+h_0}{2} - h_1, 1\right), 0\right)$.

Computation of Approximate Satisfaction Probabilities. Suppose that at the end of day 0, we have $R_0 = 0, H_0 = 0.5$, and a travel agent wants to know the risk of there being 2 consecutive days of rain over the next three days.

This probability can be computed algorithmically according to $\mathscr{M}^{\mathcal{A}}$, and for $N = 1000$, this is 0.365437 (see Appendix G of [4]). Since point $(0,500) \in S^{\mathcal{A}}$ is 0.001-bisimilar with $(0,0.5) \in S^{\mathcal{C}}$, this means that according $\mathscr{M}^{\mathcal{C}}$ with initial state $(0,0.5) \in S^{\mathcal{C}}$, the probability of there being two consecutive days of rain over the next three days is 0.365437 ± 0.003.

Analytical Validation of the Result. In this setup it is possible to evaluate the exact result for $\mathscr{M}^{\mathcal{C}}$ analytically:

$$\mathbb{P}[R_1 = 1, R_2 = 1] = \mathbb{P}[R_1 = 1] \int_0^1 f_{H_1|R_1=1}(h_1) \cdot \mathbb{P}[R_2 = 1 \mid H_1 = h_1, R_1 = 1] \, \mathrm{d}h_1,$$

which amounts to 0.199219, and similarly $\mathbb{P}[R_1 = 0, R_2 = 1, R_3 = 1] = 0.166626$. This yields

$$\begin{aligned} &\mathbb{P}[\text{Two consecutive rainy days over next 3 days}] \\ &\quad = \mathbb{P}[R_1 = 1, R_2 = 1z] + \mathbb{P}[R_1 = 0, R_2 = 1, R_3 = 1] = 0.365845, \end{aligned}$$

which is within the error bounds guaranteed by ε-trace equivalence, as expected.

7 Other Notions of ε-Bisimulation

The following condition appears in literature [1,3,14] as the definition of approximate probabilistic bisimulation. For simplicity, let us restrict our attention to finite state spaces.

Definition 9 (Alternative notion of approximate probabilistic bisimulation, adapted from [3]). *Let $\mathscr{M} = (S, \Sigma, \tau, L)$ be a LMC, where S is finite and $\Sigma = \mathcal{P}(S)$. For $\varepsilon \in [0, 1]$, a binary relation R_ε on S satisfies Definition 9 if:*

$$\forall (s_1, s_2) \in R_\varepsilon, \quad \text{we have } L(s_1) = L(s_2),$$

$$\forall (s_1, s_2) \in R_\varepsilon, \ \forall \tilde{T} \subseteq S \text{ s.t. } \tilde{T} \text{ is } R_\varepsilon\text{-closed}, \quad \text{we have } \left|\tau(s_1, \tilde{T}) - \tau(s_2, \tilde{T})\right| \leq \varepsilon.$$

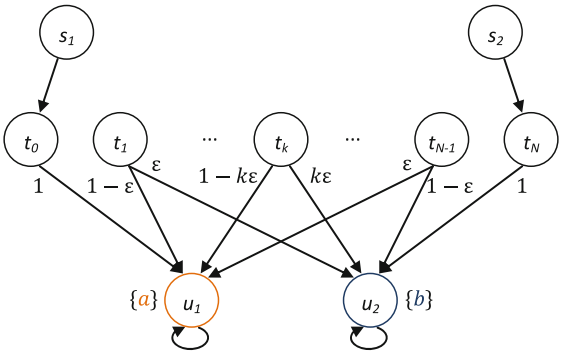
This is different from our notion of approximate probabilistic bisimulation because $\tilde{T}$ ranges over R_ε-closed sets rather than all (measurable) sets. This definition is closer to exact probabilistic bisimulation (cf. Definition 3), but it is too weak to effectively bound probabilistic trace distance.

Theorem 10. *For any $\varepsilon > 0$, there exists an LMC $\mathscr{M} = (S, \Sigma, \tau, L)$, a binary relation R_ε on S, and a pair of states $(s_1, s_2) \in R_\varepsilon$, such that R_ε satisfies the conditions in Definition 9, but the 2-step reachability probabilities from s_1, s_2 differ by 1 for some destination states.*

Proof. For $\varepsilon > 0$, let $N \in \mathbb{Z}^+$, $1/N \leq \varepsilon$. Consider the following LMC. Let $R_\varepsilon := \{(s_1, s_2)\} \cup \{(t_k, t_{k+1}) \mid k \in \{0, \cdots, N-1\}\}$.

The only R_ε-closed sets are $\{s_1, s_2\}$, $\{t_0, \ldots, t_N\}$, $\{u_1\}$, $\{u_2\}$ and unions of these sets, and so R_ε satisfies Definition 9.

We have $s_1 R_\varepsilon s_2$, and yet $P_2(s_1, \Diamond^{\leq 2} a) = 1$ but $P_2(s_2, \Diamond^{\leq 2} a) = 0$, where $\Diamond^{\leq 2} a$ is the set of length 3 traces that reach a state labelled with a.



As shown in [14] however, there is still some relationship between the probabilities of specific traces, which hinges on additional details of the structure of the transition kernel.

8 Conclusions and Extensions

In this paper we have developed a theory of $f(k)$-trace equivalence. We derived the minimum $f(k)$ such that ε-bisimulation implies $f(k)$-trace equivalence, thus extending the well known result for the exact non-probabilistic case. By linking error bounds on the total variation of length k traces to a notion of approximation based on the underlying transition kernel, we provided a means of computing upper bounds for the total variation and of synthesising abstract models with arbitrarily small total variation to a given concrete model.

It is of interest to extend our results to allow the states of the LMC to be labelled with bounded real-valued rewards, and then to limit the difference in expected reward between approximately bisimilar states.

Acknowledgments. The authors would like to thank Marta Kwiatkowska for discussions on an earlier version of this draft.

References

1. Abate, A.: Approximation metrics based on probabilistic bisimulations for general state-space Markov processes: a survey. Electron. Notes Theor. Comput. Sci. **297**, 3–25 (2013)
2. Abate, A., Katoen, J.P., Lygeros, J., Prandini, M.: Approximate model checking of stochastic hybrid systems. Eur. J. Control **16**(6), 624–641 (2010)
3. Abate, A., Kwiatkowska, M., Norman, G., Parker, D.: Probabilistic model checking of labelled Markov processes via finite approximate bisimulations. In: Breugel, F., Kashefi, E., Palamidessi, C., Rutten, J. (eds.) Horizons of the Mind. A Tribute to Prakash Panangaden. LNCS, vol. 8464, pp. 40–58. Springer, Cham (2014). doi:10.1007/978-3-319-06880-0_2

4. Bian, G., Abate, A.: On the relationship between bisimulation and trace equivalence in an approximate probabilistic context (extended version) (2017). http://arxiv.org/abs/1701.04547
5. Bollobás, B., Varopoulos, N.: Representation of systems of measurable sets. Math. Proc. Cambridge Philos. Soc. **78**(02), 323–325 (1975)
6. Buchholz, P.: Exact and ordinary lumpability in finite Markov chains. J. Appl. Probab. **31**, 59–75 (1994)
7. Castro, P.S., Panangaden, P., Precup, D.: Notions of state equivalence under partial observability. In: IJCAI (2009)
8. Chen, D., Breugel, F., Worrell, J.: On the complexity of computing probabilistic bisimilarity. In: Birkedal, L. (ed.) FoSSaCS 2012. LNCS, vol. 7213, pp. 437–451. Springer, Heidelberg (2012). doi:10.1007/978-3-642-28729-9_29
9. Chen, T., Kiefer, S.: On the total variation distance of labelled Markov chains. In: Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), p. 33. ACM (2014)
10. Comanici, G., Panangaden, P., Precup, D.: On-the-fly algorithms for bisimulation metrics. In: 2012 Ninth International Conference on Quantitative Evaluation of Systems (QEST), pp. 94–103. IEEE (2012)
11. Daca, P., Henzinger, T.A., Kretínský, J., Petrov, T.: Linear distances between markov chains. In: 27th International Conference on Concurrency Theory, CONCUR 2016, Québec City, Canada, 23–26 August 2016, pp. 20:1–20:15 (2016)
12. Desharnais, J., Edalat, A., Panangaden, P.: Bisimulation for labelled Markov processes. Inf. Comput. **179**(2), 163–193 (2002)
13. Desharnais, J., Laviolette, F., Tracol, M.: Approximate analysis of probabilistic processes: logic, simulation and games. In: Fifth International Conference on Quantitative Evaluation of Systems, QEST 2008, pp. 264–273. IEEE (2008)
14. D'Innocenzo, A., Abate, A., Katoen, J.P.: Robust PCTL model checking. In: Proceedings of the 15th ACM International Conference on Hybrid Systems: Computation and Control, pp. 275–286. ACM (2012)
15. Durrett, R.: Probability: Theory and Examples, 3rd edn. Duxbury Press, Pacific Grove (2004)
16. Esmaeil Zadeh Soudjani, S., Abate, A.: Adaptive and sequential gridding procedures for the abstraction and verification of stochastic processes. SIAM J. Appl. Dyn. Syst. **12**(2), 921–956 (2013)
17. Gebler, D., Tini, S.: Compositionality of approximate bisimulation for probabilistic systems. In: EXPRESS/SOS 2013, pp. 32–46 (2013)
18. Gibbs, A.L., Su, F.E.: On choosing and bounding probability metrics. Int. Stat. Rev. **70**(3), 419–435 (2002)
19. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). doi:10.1007/978-3-642-22110-1_47
20. Larsen, K.G., Skou, A.: Bisimulation through probabilistic testing. In: Conference Record of the 16th ACM Symposium on Principles of Programming Languages, POPL 1989, pp. 344–352 (1989)
21. Segala, R., Lynch, N.: Probabilistic simulations for probabilistic processes. Nord. J. Comput. **2**(2), 250–273 (1995)
22. Spears, W.M.: A compression algorithm for probability transition matrices. SIAM J. Matrix Anal. Appl. **20**(1), 60–77 (1998)
23. Sproston, J., Donatelli, S.: Backward bisimulation in Markov chain model checking. IEEE Trans. Softw. Eng. **32**(8), 531–546 (2006)

24. Tang, Q., van Breugel, F.: Computing probabilistic bisimilarity distances viapolicy iteration. In: 27th International Conference on Concurrency Theory, CONCUR 2016, 23-26 August 2016, Québec City, Canada, pp. 22:1–22:15 (2016)
25. Tkachev, I., Abate, A.: Formula-free finite abstractions for linear temporal verification of stochastic hybrid systems. In: Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control, pp. 283–292. ACM (2013)
26. Wu, H., Noé, F.: Probability distance based compression of hidden Markov models. Multiscale Model. Simul. **8**(5), 1838–1861 (2010)

Computing Continuous-Time Markov Chains as Transformers of Unbounded Observables

Vincent Danos[1,3], Tobias Heindel[2], Ilias Garnier[1,3], and Jakob Grue Simonsen[2(✉)]

[1] Département d'Informatique, École Normale Supérieure, Paris, France
{danos,garnier}@di.ens.fr
[2] Department of Computer Science, University of Copenhagen, Copenhagen, Denmark
{tohe,simonsen}@di.ku.dk
[3] School of Informatics, University of Edinburgh, Edinburgh, UK

Abstract. The paper studies continuous-time Markov chains (CTMCs) as transformers of real-valued functions on their state space, considered as generalised predicates and called *observables*. Markov chains are assumed to take values in a *countable* state space $\mathbf{S}$; observables $f\colon \mathbf{S} \to \mathbb{R}$ may be *unbounded*. The interpretation of CTMCs as transformers of observables is via their transition function P_t: each observable f is mapped to the observable $P_t f$ that, in turn, maps each state x to the mean value of f at time t conditioned on being in state x at time 0.

The first result is computability of the time evolution of observables, i.e., maps of the form $(t, f) \mapsto P_t f$, under conditions that imply existence of a Banach sequence space of observables on which the transition function P_t of a fixed CTMC induces a family of bounded linear operators that vary continuously in time (w.r.t. the usual topology on bounded operators). The second result is PTIME-computability of the projections $t \mapsto (P_t f)(x)$, for each state x, provided that the rate matrix of the CTMC X_t is locally algebraic on a subspace containing the observable f.

The results are flexible enough to accommodate unbounded observables; explicit examples feature the token counts in stochastic Petri nets and sub-string occurrences of stochastic string rewriting systems. The results provide a functional analytic alternative to Monte Carlo simulation as test bed for mean-field approximations, moment closure, and similar techniques that are fast, but lack absolute error guarantees.

1 Introduction

Stochastic processes are currently a very active research topic in computer science and they have been studied avidly in mathematics, even prior to

T. Heindel gratefully acknowledges support from RUBYX (project ID 628877 funded by FP7-PEOPLE).

I. Garnier is supported by the ERC project RULE (grant number 320823).

J.G. Simonsen is partially supported by the Danish Council for Independent Research Sapere Aude grant *Complexity via Logic and Algebra* (COLA).

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 338–354, 2017.
DOI: 10.1007/978-3-662-54458-7_20

Kolmogorov's axiomatic approach to probability. For the special case of continuous-time Markov chains (CTMCs), we shall study *how they act* on functions from their state space to the reals, which we call *observables*, alluding to measurement of observable quantities of states. In analogy to predicate transformer semantics [Koz83], observables are considered as generalised predicates that Markov chains transform over time, thus leading to observations that evolve continuously in time. The principal question is *how to compute* the time evolution of observables.

On computability of time-dependent observations. The following scenario introduces the basic concepts and leads to the core questions of computability. Suppose, we want to compute the mean $\mathsf{E}[f(X_t)]$ of an observable f on a CTMC X_t with denumerable state space. However, initially, we are given only a specification of its dynamics, say, by a finite model that determines the *transition function* P_t, i.e., the matrix of probabilities $p_{t,xy}$ to jump from state x to state y during a time interval of length t; the initial distribution, i.e., the distribution π of X_0, will be available only much later.

As we do not know the initial distribution in advance, we want to split the computation of the mean $\mathsf{E}[f(X_t)] = \sum_{x,y} \pi(x) p_{t,xy} f(y)$ into a first phase in which we compute conditional means $\mathsf{E}_x(f(X_t)) = \mathsf{E}[f(X_t) \mid X_0 = x] = \sum_y p_{t,xy} f(y)$ for a sufficiently large, but finite set of possible initial states x and a second phase for integration w.r.t. the initial distribution π. The two core questions for an approximation of the mean $\mathsf{E}[f(X_t)]$ to desired precision $\epsilon > 0$ using the described two phase approach are: Is it actually possible to restrict to a finite set of initial states x that we need to consider? If so, can we compute the conditional means $\mathsf{E}_x(f(X_t)) = \sum_y p_{t,xy} f(y)$ w.r.t. these states to sufficient precision?

We want to condense these two questions into a single computability question. For this, we first congregate the conditional means $\mathsf{E}_x(f(X_t))$ into a single observable $P_t f$ with $P_t f(x) = \mathsf{E}_x(f(X_t))$; then, fixing a suitable Banach space of observables, it makes sense to ask for an approximation of $P_t f$ to precision ϵ. Finally, employing the framework of type 2 theory of effectivity, we can simply ask if the observable $P_t f$ is computable. We go one step further and study computability of the time evolution of these observables.

Examples. For motivation, we give two paradigmatic examples of *transient means* of the form $\mathsf{E}[f(X_t)]$. The first example of a stochastic process of the form $f(X_t)$, i.e., a pair of a CTMC X_t and an observable f, is the classic CTMC model of a set of chemical reactions where states are multisets over a finite set of species with the count of a certain chemical species as observable; thus, we are interested in the time evolution of the mean count of a certain species.

An example native to computer science is the stochastic interpretation of any string rewriting system as a CTMC X_t. An obvious class of observables for string rewriting are functions that count the occurrence of a certain word as substring in each state of the CTMC X_t; note that this is different from counting "molecules" as there is always only a single word! For example, consider the

string rewriting system with the single rule $a \to aba$ and initial state a: the mean occurrence count of the letter a grows as the exponential function e^t while the mean occurrence count of the word aa is zero at all times; adding the rule $ba \to ab$ does not change the mean a-count but renders the mean count of the word aa non-trivial.

Note that these two classes of models are only the most basic types of rule-based models, besides more powerful examples such as Kappa models [DFF+10] and stochastic graph transformation [HLM06].[1] The results of this paper are independent of any particular modelling language for CTMCs.

Finite state case. For the sake of clarity, let us describe explicitly the objects that we would manipulate and compute in the basic case where the CTMC has a finite state space. The dynamics of a continuous-time Markov chain on a finite state space $\mathbf{S}$ is entirely captured by its q-matrix, which is an $\mathbf{S} \times \mathbf{S}$-indexed real matrix in which every row sums to zero and all negative entries lie on the diagonal. Every q-matrix Q induces a matrix semigroup $t \mapsto P_t = e^{tQ}$ which is exactly the transition function of the CTMC where e^{tQ} is the matrix exponential of tQ. Viewing a distribution π on $\mathbf{S}$ as a row vector, the map $t \mapsto \pi P_t$ describes the time evolution of the distribution over states at time t starting from the initial distribution π at time 0. In particular, if X_0 is distributed according to π, the associated CTMC X_t is distributed according to πP_t. Dually, for any function $f\colon \mathbf{S} \to \mathbb{R}$ (seen as a column vector), $P_t f$ is the vector of conditional means $\mathsf{E}_x(f(X_t))$ of f at time t as a function of the initial state x. As said before, the time evolution of observables in the CTMC with q-matrix Q, i.e., the map $t \mapsto P_t f$, is the main object of interest for the present paper; the principal question is whether it is computable.

For the finite state case, computability of $t \mapsto P_t f$ is trivial, assuming f is computable and the q-matrix consists of rational entries. Here, computability is in the sense of type 2 theory of effectivity (TTE), which for the function $t \mapsto P_t f$ means that there are approximation schemes for all coordinates $P_t f(x)$ to arbitrary desired precision. Even the whole matrix P_t is computable, as it is finite dimensional in the finite state case; finally, observe that all observables on a finite state space are necessarily bounded functions.

The general case. The characterisation of the function $t \mapsto P_t f$ as the unique solution to the initial value problem (IVP)

$$\begin{aligned} \tfrac{\mathrm{d}}{\mathrm{d}t} u_t &= Q u_t \\ u_0 &= f \end{aligned} \tag{1}$$

will turn out to be very useful as it generalises rather naturally to arbitrary Banach spaces [Ein52]. However, there are two points to note. While the q-matrix Q is a bounded linear operator on the finite-dimensional vector space of all

[1] In fact, stochastic string rewriting is the restriction of stochastic graph transformation [HLM06] to directed, connected, acyclic, edge labelled graphs with in and out degree of nodes bounded by one, i.e., to graphs consisting of a unique maximal path.

observables when we have a finite state space, this does not hold true for the general case. Moreover, the observable f itself might be *unbounded*, which poses an additional difficulty for solving an IVP like (1) as described in Sect. 3.3.

The first contribution of the paper consists in setting up a suitable generalisation of the initial value problem (1) in a Banach space space of observables such that the time-dependent observable $P_t f$, mapping a state x to the conditional mean $\mathsf{E}_x(f(X_t))$, is its unique solution; for this, we heavily use the functional analytic techniques recently developed in Refs. [Spi12,Spi15]. The main contribution is Theorem 2 on computability of the function $(t, f) \mapsto P_t f$ under mild additional assumptions on the q-matrix of the Markov chain, putting to use recent results by Weihrauch and Zhong [WZ07] on computability of solutions of initial value problems in Banach spaces. The sufficient conditions are general enough to encompass many interesting unbounded observables. Finally, we show that, for a fixed state x and observable f, the time evolution of the conditional mean $\mathsf{E}_x(f(X_t))$ is PTIME computable (Theorem 3) under conditions that are strict enough to re-use results on linear ODEs [PG16], yet general enough to capture mean word counts in context-free stochastic string rewriting (Corollary 2).

Related work. Computability of continuous-time Markov chains as transformers of unbounded observables is related to computation of transient means $\mathsf{E}[f(X_t)]$ of an observable f on a CTMC X_t with countable state space. Computability of transient means, in turn, is related to first passage probabilities of a decidable set of states U (cf. [GM84, Sect. 6.2]): the latter problem can be reduced to computing transient means by use of an indicator function that checks for states in U and a modified dynamics of the Markov chain, disabling jumps out of U.

Adaptive uniformisation (AU) [VMS93,VMS94] allows one to compute transient means of bounded observables without further complications. However, AU requires the initial distribution to be finite and known from the start. Our results are not subject to these two restrictions, though we need that for each desired precision ϵ, there is a finite number of states to which we can restrict possible initial distributions, which is a restriction on the dynamics of the CTMC.[2] The main novelties are the focus on the observable and its time evolution, answering the question of how the dynamics a Markov chain acts on an observable, in general and independent of the initial distribution, and its computability to arbitrary desired precision. We even treat the case of unbounded observables, relying on recent mathematical results [Spi12,Spi15].

Model-checking of continuous-time Markov chains typically concern properties of sample paths of CTMCs relative to a labeling function on states [BHHK03]. In the present paper we neither have a labeling function nor do we rely on sample paths, explicitly. However, it may be that the methods of the present paper can be adapted to the labeled case.

Structure of the paper. The paper starts out with the detailed description of the motivating examples, namely string rewriting and stochastic Petri nets. Then

[2] Specifically, all CTMCs that fail to be Feller processes [RR72] are problematic.

we review the mathematical preliminaries, in particular continuous-time Markov chains on a countable state space and the basic concepts of transition functions and q-matrices. The generalisation of the initial value problem (1) and the characterisation of the continuous-time observation transformation $t \mapsto P_t f$ of an observable f by the transition function P_t of a CTMC (Theorem 1) are given in Sect. 4. The main result (Theorem 2) on computability of the continuous-time transformation of observables by CTMCs is presented in Sect. 5. In Sect. 6, we show PTIME-computability of the time evolution of the conditional mean $\mathsf{E}_x(f(X_t))$, for all states x, under assumptions that allow to restrict to a finite-dimensional space (Theorem 3) and its direct consequence for string rewriting (Corollary 2). Finally, we conclude with a summary of results and directions for future work.

2 Two Motivating Examples of CTMCs with Observables

We illustrate our constructions with: (i) chemical reaction networks (CRN), aka stochastic Petri nets, and (ii) stochastic string rewriting as a simple example of (rule-based) modelling. In both cases, the construction of the q-matrix implied by a model is readily done, and so is the definition of a natural set of *unbounded* observables with clear relevance to the dynamics of a model: word occurrence counts for stochastic string rewriting (Definition 2) and multiset inclusions for Petri nets (Definition 3).

2.1 Stochastic String Rewriting and Word Occurrences

Stochastic string rewriting can be thought of as never ending, fair competition between all redexes of rules, "racing" for reduction; the formal definition is as follows, in perfect analogy to Ref. [HLM06] which covers the case of graphs.

Definition 1 (Stochastic string rewriting). *Let $\rho = l \to r \in \Sigma^+ \times \Sigma^+$ be a rule. The* q-matrix of ρ, *denoted by Q_ρ, is the q-matrix $Q_\rho = (q^\rho_{uv})_{u,v\in\Sigma^+}$ on the state space of words Σ^+ with off-diagonal entries*

$$q^\rho_{uv} = \left|\left\{(w, w') \in \Sigma^* \times \Sigma^* \;\middle|\; u = wlw',\ v = wrw'\right\}\right|$$

for each pair of words $u, v \in \Sigma^+$ such that $u \neq v$, and diagonal entries $q^\rho_{uu} = -\sum_{v\neq u} q^\rho_{uv}$ for all $u \in \Sigma^+$. For a finite set of rules $\mathcal{R} \subseteq \Sigma^+ \times \Sigma^+$, we define $Q_\mathcal{R} = \sum_{\rho\in\mathcal{R}} Q_\rho$, and with additional choices of rate constants $\mathsf{k}\colon \mathcal{R} \to \mathbb{Q}^+$, we define $Q_{\mathcal{R},\mathsf{k}} = \sum_{\rho\in\mathcal{R}} \mathsf{k}_\rho Q_\rho$.

For a given rule set $\mathcal{R}$, each entry $q^\mathcal{R}_{uv}$ of the q-matrix corresponds to the propensity to rewrite: it is just the number of ways in which u can be rewritten to v. We shall usually work without rate constants for the sake of readability. Note that the use of Σ^+ for the left and right hand side of rules is convenient to get string rewriting as a special case of graph transformation in a straightforward manner.

The occurrence counting function of a word as sub-string in the state of the CTMC of $\mathcal{R}$ is as follows.

Definition 2 (Word counting functions). *Let $w \in \Sigma^+$ be a word. The* w-counting function*, denoted by $\sharp_w \colon \Sigma^+ \to \mathbb{R}_{\geq 0}$, maps each word $x \in \Sigma^+$ to $\sharp_w(x) = |\{(u, v) \in \Sigma^* \times \Sigma^* \mid x = uwv\}|$.*

2.2 Stochastic Petri Nets and Sub-multiset Occurrences

We recall the definition of stochastic Petri nets and occurrence counting of a multisets. Note that for the purposes of the present paper, places and species are synonymous.

Definition 3 (Multisets and multiset occurrences). *A* multiset *over a finite set $\mathcal{P}$ of* places *is a function $x \colon \mathcal{P} \to \mathbb{N}$ that maps each place to the number of* tokens *in that place. Given a multiset, $x \in \mathbb{N}^{\mathcal{P}}$, the* x-occurrence counting *function $\sharp_x \colon \mathbb{N}^{\mathcal{P}} \to \mathbb{N}$ is defined by*

$$\sharp_x(y) = \begin{cases} \frac{y!}{(y-x)!} & x \leq y \\ 0 & \textit{otherwise} \end{cases}$$

where $z! = \prod_{p \in \mathcal{P}} z(p)!$ is the multiset factorial for all $z \in \mathbb{N}^{\mathcal{P}}$.

Definition 4 (Stochastic Petri net). *Let $\mathcal{P}$ be a finite set of* places*. A* stochastic Petri net *over $\mathcal{P}$ is a set*

$$\mathcal{T} \subseteq \mathbb{N}^{\mathcal{P}} \times \mathbb{R}_{>0} \times \mathbb{N}^{\mathcal{P}}$$

where $\mathbb{N}^{\mathcal{P}}$ is the set of multisets over $\mathcal{P}$, which are called markings *of the net; elements of the set $\mathcal{T}$ are called* transitions*. The q-matrix $Q_{l,k,r}$ on the set of markings for a transition $(l, k, r) \equiv l \to^k r \in \mathcal{T}$ has off-diagonal entries*

$$q_{xy}^{l,k,r} = \begin{cases} k \cdot \sharp_l(x) & \sharp_l(x) > 0, y = x - l + r \\ 0 & \textit{otherwise} \end{cases}$$

where addition and subtraction is extended pointwise to $\mathbb{N}^{\mathcal{P}}$. The q-matrix of $\mathcal{T}$ is $Q_{\mathcal{T}} = \sum_{(l,k,r) \in \mathcal{T}} Q_{l,k,r}$.

3 Preliminaries

For the remainder of the paper, we fix an at most countable set **S** as state space.

3.1 Transition Functions and q-Matrices

We first recall the basic definitions of transition functions and q-matrices. We make the usual assumptions [And91] one needs to work comfortably: namely that q-matrices are stable and conservative and that transition functions are standard and also minimal as described at the end of Sect. 3.1.

With these assumptions in place, transition functions and q-matrices determine each other, and one can freely work with one or the other as is most convenient.

Definition 5 (Standard transition function [And91, p. 5f]). *A* transition function *on* $\mathbf{S}$ *is a family* $\{P_t\}_{t\in\mathbb{R}_{\geq 0}}$ *of* $\mathbf{S}\times\mathbf{S}$*-matrices* $P_t = (p_{t,xy})_{x,y\in\mathbf{S}}$ *with non-negative, real entries* $p_{t,xy}$ *such that*

1. $\lim_{t\searrow 0} p_{t,xx} = 1$ *for all* $x \in \mathbf{S}$*;*
2. $\lim_{t\searrow 0} p_{t,xy} = 0$ *for all* $x, y \in \mathbf{S}$ *such that* $y \neq x$*;*
3. $P_{t+s} = P_t P_s = (\sum_{z\in\mathbf{S}} p_{s,xz} p_{t,zy})_{x,y\in\mathbf{S}}$ *for all* $s, t \in \mathbb{R}_{\geq 0}$*; and*
4. $\sum_{z\in\mathbf{S}} p_{t,xz} \leq 1$ *for all* $x \in \mathbf{S}$ *and* $t \in \mathbb{R}_{\geq 0}$.

Thus, each row of a transition function corresponds to a sub-probability measure, and transition functions converge entry-wise to the identity matrix at time zero.

Taking entry-wise derivatives of a transition function at time 0 is possible [Kol51, Aus55] and gives a q-matrix.

Definition 6 (q-matrix). *A* q-matrix *on* $\mathbf{S}$ *is an* $\mathbf{S}\times\mathbf{S}$*-matrix* $Q = (q_{xy})_{x,y\in\mathbf{S}}$ *with real entries* q_{xy} *such that* $q_{xy} \geq 0$ *(if* $x \neq y$*),* $q_{xx} \leq 0$*, and* $\sum_{z\in\mathbf{S}} q_{xz} = 0$ *for all* $x, y \in \mathbf{S}$.

Conversely, for each q-matrix, there exists a unique entry-wise minimal transition function that solves Eq. (2) [And91, Theorem 2.2],

$$\frac{\mathrm{d}}{\mathrm{d}t} P_t = Q P_t,\ P_0 = I \tag{2}$$

which is called *the* transition function of Q. From now on, we assume that all transition functions are minimal solutions to Eq. (2) for some q-matrix Q (see [Nor98, p. 69]).

3.2 The Abstract Cauchy Problem for $P_t f$

Abstract Cauchy problems (ACPs) in Banach spaces [Ein52] are the classic generalisation of finite-dimensional initial value problems (see also Refs. [ABHN11, EN00]). Specifically, we want to obtain $P_t f$ as unique solution u_t of the following generalisation of our earlier IVP (1):

$$\begin{aligned} \tfrac{\mathrm{d}}{\mathrm{d}t} u_t &= \mathcal{Q} u_t \quad (t \geq 0) \\ u_0 &= f \end{aligned} \tag{ACP}$$

where f is an observable and $\mathcal{Q}$ is a linear operator which plays the role of the q-matrix. ACPs that allow for unique differentiable solutions are intimately related to strongly continuous semigroups (SCSGs) and their generators (see, e.g., [EN00, Proposition II.6.2]).

Definition 7. *Let* $\mathcal{B}$ *be a real Banach space with norm* $\|\_\|$*. A* strongly continuous semigroup *on* $\mathcal{B}$ *is a family* $\{P_t\}_{t\in\mathbb{R}_{\geq 0}}$ *of bounded linear operators* P_t *on* $\mathcal{B}$ *satisfying (i)* $P_0 = I_{\mathcal{B}}$ *(the identity on* $\mathcal{B}$*); (ii)* $P_{t+s} = P_t P_s$*, for all* $s, t \in \mathbb{R}_{\geq 0}$*; and (iii)* $\lim_{h\searrow 0} \|P_h f - f\| = 0$*, for all* $f \in \mathcal{B}$*. The* infinitesimal generator $\mathcal{Q}$ *of a strongly continuous semigroup* P_t *on* $\mathcal{B}$ *is the linear operator defined by* $\mathcal{Q} f = \lim_{h\searrow 0} {}^1\!/_h (P_h f - f)$ *for all* $f \in \mathcal{B}$ *that belong to the* domain of definition $\mathrm{dom}(\mathcal{Q}) = \left\{ f \in \mathcal{B} \;\middle|\; \text{The limit } \lim_{h\searrow 0} {}^1\!/_h (P_h f - f) \text{ exists.} \right\}$.

There are a few points worth noting on how to pass from the IVP (1) to a corresponding ACP. First, the topological vector space of *all* observables $\mathbb{R}^{\mathbf{S}}$ cannot be equipped with a suitable complete norm to turn it into a Banach space. Therefore, one has to look for a subspace $\mathcal{B} \subset \mathbb{R}^{\mathbf{S}}$ wherein to interpret the above equation. Second, as $P_t f = u_t$ is the desired solution, and $P_0 = I$, it follows that $\frac{\mathrm{d}}{\mathrm{d}t} P_t f|_{t=0} = \mathcal{Q}f$. If this derivative does not exist, $\mathcal{Q}f$ is simply not defined. In fact, as is clear from the examples in Sect. 2, we can only expect $\mathcal{Q}$ to be partially defined as it is not a bounded operator, in general.[3]

On the positive side, if we know that P_t is an SCSG on $\mathcal{B}$, meaning $\lim_{h \searrow 0} P_h f = f$ for all $f \in \mathcal{B}$, we can take $\mathcal{Q}$ to be its generator, i.e., the linear operator defined on $g \in \mathcal{B}$ by $\mathcal{Q}g := \frac{\mathrm{d}}{\mathrm{d}t} P_t g|_{t=0}$ whenever this limit exists, and obtain $P_t f$ as *unique* solution of (ACP) [EN00, Proposition II.6.2]. Even better, in this case, not only does (ACP) have $P_t f$ as unique solution, but we get an explicit approximation scheme:

$$P_t f = \lim_{n \to \infty} e^{tA_n} f \tag{3}$$

where θ is a constant of the SCSG such that $nI - \mathcal{Q}$ is invertible for $n > \theta$ and the operators $A_n = n\mathcal{Q}(nI - \mathcal{Q})^{-1}$, known as *Yosida approximants*, are bounded.

Yosida approximants are the cornerstone of the generation theorems for SCSGs [EN00, Corollary 3.6] that allow one to pass from the generator $\mathcal{Q}$ to the corresponding SCSG. The constant θ also bounds the growth of the SCSG in norm, namely $\|P_t\| \leq Me^{\theta t}$ for some M. This should already make clear that Eq. (3) is crucial to obtain error bounds for results on the computability of SCSGs. In fact, it is the starting point of the proof of the main result on the computability of SCSGs [WZ07, Theorem 5.4.2, p. 521].

It remains to see whether we can exhibit Banach spaces to build ACPs that accomodate interesting (specifically unbounded) observables.

3.3 Banach Space Wanted!

Table 1 gives an overview of initial value problems for transient distributions (first row) and transient conditional means (second row). Transient distributions are summable sequences, and transition functions form SCSGs [Reu57] and therefore allow for a well-posed corresponding ACP. But the classic example of a Banach space to reason about conditional means [RR72] is the space $C_0(\mathbf{S})$ of functions vanishing at infinity, i.e., functions $f \colon \mathbf{S} \to \mathbb{R}$ such that for all $\epsilon > 0$, the set $\{x \in \mathbf{S} \mid f(x) \geq \epsilon\}$ is finite, equipped with the supremum norm. The corresponding processes are called Feller transition functions [And91, Sect. 1.5] and verify a principle of finite velocity of information flow (for all t, y, the function $x \mapsto p_{t,xy}$ vanishes as x goes to infinity).

[3] Even when $\mathcal{Q}f$ is defined, one has to check $\mathcal{Q}f = Qf$, that is to say: $1/h(P_h f - f)$ converges to Qf in the Banach space norm. But this will turn out to be easy compared to finding sufficient conditions for $\mathcal{Q}f$ to be defined.

Table 1. Transition functions acting on Banach spaces: state of the art

		solution (finite $\mathbf{S}$)	generalisation (countably infinite $\mathbf{S}$)	
IVP transient distributions	$\frac{\mathrm{d}}{\mathrm{d}t}\pi_t = \pi_t Q$, $\pi_0 = \pi$	$\pi_t = \pi e^{Qt}$	$\pi_t = \pi P_t$, P_t SCSG on $L^1(\mathbf{S})$, in general	
IVP transient conditional means	$\frac{\mathrm{d}}{\mathrm{d}t}u_t = Qu_t$, $u_0 = f$	$u_t = e^{Qt}f$	$\sup_{i\in\mathbf{S}} -q_{ii} < \infty$ or Feller	$\sup_{i\in\mathbf{S}} -q_{ii} = \infty$ not Feller
			$u_t = P_t f$, P_t SCSG on $L^\infty(\mathbf{S})$ or $C_0(\mathbf{S})$	[Spi12, Theorem 6.3] or open problem

4 Spieksma's Theorem

A solution is provided by a result of Spieksma [Spi12, Theorem 6.3], giving a class of candidate Banach spaces $\mathcal{B}$ for a given q-matrix Q and an observable f of interest such that P_t forms an SCSG on $\mathcal{B}$ (Theorem 1.1). As a consequence, we are led to ACPs generalising the IVP (1) in which the operator $\mathcal{Q}$ is the generator of the transition function P_t (seen as an SCSG on $\mathcal{B}$) and is a restriction of the q-matrix Q, i.e., $\mathcal{Q}f = Qf$ for all $f \in \mathrm{dom}(\mathcal{Q})$. Moreover, we obtain a characterization of part of the the domain $\mathrm{dom}(\mathcal{Q})$ (Proposition 1). The results of this section set the mathematical stage for the main results.

4.1 Weighted C_0-Spaces and Drift Functions

The Banach spaces that we shall work with are weighted variants of $C_0(\mathbf{S})$ such that functions vanish at infinity relative to a chosen weight function on states.

Definition 8 (Weighted $C_0(\mathbf{S})$-spaces). *Let $\mathbf{S}$ be a set and let $W\colon \mathbf{S} \to \mathbb{R}_{>0}$ be a positive real-valued function, referred to as a* weight. *The Banach space $C_0(\mathbf{S}, W)$ consists of functions $f\colon \mathbf{S} \to \mathbb{R}$ such that f/W vanishes at infinity, where $(f/W)(x) = f(x)/W(x)$. The norm $\|\_\|_W$ on such functions f is $\|f\|_W = \sup_{x\in\mathbf{S}} |f(x)/W(x)|$.*

As $C_0(\mathbf{S}, W)$ is isometric to $C_0(\mathbf{S})$ it is indeed a Banach space. It is also a closed subspace of $L^\infty(\mathbf{S}, W)$, the set of functions such that f/W is bounded. We shall use later the fact that:

Lemma 1. *Finite linear combinations of indicator functions,*[4] *form a dense subset of $C_0(\mathbf{S})$.*

Spieksma's theorem [Spi12, Theorem 6.3] will be in terms of so-called drift functions, which intuitively are functions whose mean w.r.t. a given CTMC grows with at most constant rate.

[4] The indicator function $\mathbb{1}_x$ is defined as usual as $\mathbb{1}_x(y) = \delta_{xy}$.

Definition 9 (Drift function). *Let Q be a q-matrix on $\mathbf{S}$, and let $c \in \mathbb{R}$. A function $W \colon \mathbf{S} \to \mathbb{R}_{>0}$ is called a* c-drift function *for Q if for all $x \in \mathbf{S}$ $(QW)(x) := \sum_{y \in \mathbf{S}} q_{xy} W(y) \leq cW(x)$.*

We shall say that W is a *drift function* for Q if there exists $c \in \mathbb{R}$ such that it is a c-drift function for Q. One can show that $P_t W \leq e^{ct} W$ in this case. Thus, drift functions control their own growth under the transition function.

4.2 Transition Functions as Stronlgy Continuous Semigroups

The crux of Spieksma's theorem [Spi12, Theorem 6.3] is a pair of positive drift functions V, W for Q such that $V \in C_0(\mathbf{S}, W)$, i.e., such that the quotient V/W vanishes at infinity. Intuitively, qua drift function, their growth is at most exponential in mean; moreover V is negligible compared to W at infinity, and thus functions on the order of V are as good as functions vanishing at infinity, in analogy to the case of Feller processes [RR72], which is exactly the class of CTMCs whose transition functions induce SCSGs on $C_0(\mathbf{S})$. Hence, the following result is a first step towards a theory of weighted Feller processes.

Theorem 1. *Let P_t be a transition function on the state space $\mathbf{S}$ with q-matrix Q and let $V, W \colon \mathbf{S} \to \mathbb{R}_{>0}$ be drift functions for Q. Then the following hold.*

1. *The transition function P_t induces an SCSG on $C_0(\mathbf{S}, W)$ iff $V \in C_0(\mathbf{S}, W)$.*
2. *If $V \in C_0(\mathbf{S}, W)$, for all $f \in C_0(\mathbf{S}, W)$ and $t \in \mathbb{R}_{\geq 0}$, $P_t f$ is given by Eq. (3) in the Banach space $C_0(\mathbf{S}, W)$ where $\mathcal{Q}$ is the generator of P_t.*

The first part of the theorem is proved in [Spi12, Theorem 6.3]; the second part follows from the general theory of ACPs. Note that f does not need to be in the domain of $\mathcal{Q}$, in which case we only obtain a *mild solution* to the ACP [EN00, Definition II.6.3], i.e., a solution to its integral form which is not everywhere differentiable. In fact, the solution is differentiable if and only if f belongs to the domain of the generator [EN00, Proposition II.6.2].

4.3 On the Domain of the Generator

One difficulty in working with SCSGs is to find a useful description of the domain of their generator. However, the graph of the infinitesimal generator of an SCSG is completely determined by the restriction to any dense subset. The following characterisation of subsets of the domains of generators of SCSGs that are obtained via Theorem 1 is a corrected weakening [Spi16] of the second part of Theorem 6.3 of Ref. [Spi12], naturally generalising the classic result for Feller processes [RR72, Theorem 5].

Proposition 1. *Let P_t be a transition function on $\mathbf{S}$ with q-matrix Q and let $V, W \colon \mathbf{S} \to \mathbb{R}_{>0}$ be positive drift functions for Q such that $V \in C_0(\mathbf{S}, W)$. Let $\mathcal{Q}$ be the generator of the SCSG P_t on $C_0(\mathbf{S}, W)$ (cf. Theorem 1).*

For all $f \in C_0(\mathbf{S}, W)$ *that satisfy* $\|f\|_V < \infty$ *and* $Qf \in \mathrm{dom}(Q)$*, we have*

$$Qf = \mathcal{Q}f = \lim_{h \searrow 0} {}^1\!/\!_h (P_t f - f), \tag{4}$$

i.e., the latter limit exists in $C_0(\mathbf{S}, W)$ *and in particular* $f \in \mathrm{dom}(\mathcal{Q})$*.*

We have now covered the mathematical ground needed to characterise the transformation of observations by transition functions of CTMCs as solutions of an ACP, generalising the finite state case of IVP (1). This, however, does not immediately yield an *algorithm* for computing transient means. Even transient conditional distributions can fail to be computable [AFR11]! Before we proceed to the question of computability, let us return to our two classes of examples.

4.4 Applications: String Rewriting and Petri Nets

We now give examples of drift functions for stochastic string rewriting and Petri nets. The former case is well-behaved since the mean letter count grows at most exponentially. The case of Petri nets will be more subtle and we shall give an example of an explosive Petri net such that we can nevertheless reason about conditional means of unbounded observables.

For string rewriting, we have canonical drift functions.

Lemma 2 (Powers of length are drift functions). *Let* $\mathcal{R} \subseteq \Sigma^+ \times \Sigma^+$ *be a finite string rewriting system and let* $n \in \mathbb{N}^+$ *be a positive natural number. There exists a constant* $c_n \in \mathbb{R}_{>0}$ *such that* $|\_|^n \colon \Sigma^+ \to \mathbb{R}_{\geq 0}$ *is a* c_n*-drift function.*

Now, we can apply Spieksma's method to get a Banach space for reasoning about conditional means and moments of word counting functions.

Corollary 1 (Stochastic string rewriting). *Let* $\mathcal{R}$ *be a finite string rewriting system, let* $n \in \mathbb{N}\backslash\{0\}$*, and let* $|\_| \colon \Sigma^+ \to \mathbb{N}$ *be the word length function. The transition function* P_t *of q-matrix* $Q_{\mathcal{R}}$ *is an SCSG on* $C_0(\Sigma^+, |\_|^n)$*.*

Thus, all higher conditional moments of word counting functions can be accommodated in a suitable Banach space. The case of Petri nets is more subtle, since, in general, the (weighted) token count is not a drift function.

Example 1. Consider the Petri net with single transition $2A \to^1 3A$ and with one place A. The token count $\sharp_A$ is not a drift function. In fact, the corresponding CTMC is explosive (by Theorem 2.1 of Ref. [Spi15]).

Our final example is an extension of the previous explosive CTMC with a new species whose count can nevertheless be treated using Theorem 1.

Example 2 (Unobserved explosion). Consider the Petri net with transitions

$$\{2A \to^1 3A, B \to^1 2B\}.$$

The underlying CTMC is explosive, and we cannot apply Theorem 1 to compute the transient conditional mean of the A-count for the exact same reason as in Example 1. However, we can do so for the B-count, using the weight function $W = \sharp_B{}^2$ and observable $f = \sharp_B$. Putting $V = f$ allows one to apply Spieksma's recipe (ruling out states with B-count 0 for convenience). The conditional mean $\mathsf{E}_{2A+B}(\sharp_B(X_t))$ can be best understood by adding a coffin state, on which both the A- and B-count are zero and in which the Markov chain resides after (the first and only) explosion.

5 Computability

We follow the school of type-2 theory of effectivity. A real number x is computable iff there is a Turing machine that on input $d \in \mathbb{N}$ (the desired precision), outputs a rational number r with $|r - x| < 2^{-d}$. Next, a function $g\colon \mathbb{R} \to \mathbb{R}$ is computable if there is a Turing machine that, for each $x \in \mathbb{R}$, takes an arbitrary Cauchy sequence with limit x as input and generates a Cauchy sequence that converges to $g(x)$—where convergence has to be sufficiently rapid, e.g., by using the dyadic representation of the reals.

Computability extends naturally to any Banach space $\mathcal{B}$ other than $\mathbb{R}$. We only need a recursively enumerable dense subset on which the norm, addition and scalar multiplication are computable, thus making $\mathcal{B}$ a *computable* Banach space; usually, the dense subset is induced by a basis of a dense subspace. For weighted C_0-spaces (with computable weight functions) and their duals $(C_0(\mathbf{S}, W))^*$, we fix an arbitrary enumeration of all *rational* linear combinations of indicator functions $\mathbb{1}_x$; for the Banach space of bounded linear operators on weighted C_0-spaces we use the standard construction for continuous function spaces [WZ07, Lemma 3.1]. A SCSG P_t is computable if the function $t \mapsto P_t$ from the reals to the Banach space of bounded linear operators is computable. The computable SCSGs correspond to those obtained from CTMCs through Theorem 1.

We restrict to row- and column-finite q-matrices with rational entries in our main result, motivated by the observation that we do not lose any of the intended applications to rule-based modelling.

Theorem 2 (Computability of CTMCs as observation transformers). *Let Q be a q-matrix on $\mathbf{S}$, let $W\colon \mathbf{S} \to \mathbb{R}_{\geq 0}$ be a positive drift function for Q such that there exists $V \in C_0(\mathbf{S}, W)$ that is a drift function for Q. If*

- *the q-matrix Q is row- and column-finite, consists of rational entries, and is computable as a function $Q\colon \mathbf{S}^2 \to \mathbb{Q}$ and the function $y \mapsto \{x \in \mathbf{S} \mid q_{xy} \neq 0\}$ is computable, and*
- *$W\colon \mathbf{S} \to \mathbb{Q}$ is computable,*

the following hold.

1. *The SCSG P_t is computable.*
2. *The evolution of conditional means $(t, f) \mapsto P_t f$ is computable as partial function from $\mathbb{R} \times C_0(\mathbf{S}, W)$ to $C_0(\mathbf{S}, W)$ defined on $\mathbb{R}_{\geq 0} \times C_0(\mathbf{S}, W)$.*
3. *The evolution of means $(\pi, t, f) \mapsto \pi P_t f$ is computable as partial function from $C_0(\mathbf{S}, W)^* \times \mathbb{R} \times C_0(\mathbf{S}, W)$ to $\mathbb{R}$ defined on $C_0(\mathbf{S}, W)^* \times \mathbb{R}_{\geq 0} \times C_0(\mathbf{S}, W)$.*

Proof. We shall apply a result by Weihrauch and Zhong on the computability of SCSGs [WZ07, Theorem 5.4]. Applying this result requires some extra information:

1. the SCSG P_t must be bounded in norm by $e^{\theta t}$ for some positive constant θ;
2. we must have a recursive enumeration of a dense subset of the graph of the infinitesimal generator of the SCSG P_t.

We first show that the constant θ, featuring in Theorem 1, i.e., the witness that W is a θ-drift function for Q, satisfies $\|P_t\| \leq e^{\theta t}$ (using the first part of the proof of Theorem 6.3 of Ref. [Spi12]). Next, we obtain a recursive enumeration of a dense subset $A \subseteq \mathrm{dom}\mathcal{Q}$ of the domain of the generator $\mathcal{Q}$ by applying Q to all rational linear combinations of indicator functions $\mathbb{1}_x$. Note that for the latter, we use that indicator functions belong to the domain of the generator and $\mathcal{Q}\mathbb{1}_x = Q\mathbb{1}_x$ by Proposition 1.

Now, by Theorem 5.4.2 of Ref. [WZ07], we obtain the first two computability results, as $(\theta, A, 1)$ is a so-called piece of type IG-information [WZ07, p. 513]. Finally, the third point amounts to showing computability of the duality pairing

$$\langle \_, \_ \rangle \colon (C_0(\mathbf{S}, W))^* \times C_0(\mathbf{S}, W) \to \mathbb{R}.$$

This theorem immediately gives computability of the CTMCs and (conditional) means for stochastic string rewriting and Petri nets discussed in Sect. 4. Note that the theorem does not assume that V itself is computable; its role is to establish that the transition functions is an SCSG, but V plays no role in the actual computation of the solution. Note also that the algorithms that compute the functions $t \mapsto P_t$, $(t, f) \mapsto P_t f$, and $(\pi, t, f) \mapsto \pi P_t f$ push the responsibility to give arbitrarily good approximations of the respective input parameters π, t and f to the user. This however is no problem for any of our examples or rule-based models in general: t is typically rational, f is computable and even to the natural numbers, and π is often finitely supported or a Gaussian.

Computability ensures existence of algorithms computing transient means, but yields no guarantees of the efficiency of such algorithms. We now proceed to a special case that (i) encompasses a number of well-known examples, including context-free string rewriting, and (ii) leads to PTIME computability, by reducing the problem of transient conditional means to solving finite linear ODEs.

6 The Finite Dimensional Case and PTIME via ODEs

We now turn to the special case where we can restrict to finite dimensional subspaces $\mathcal{B} \subseteq \mathbb{R}^{\mathbf{S}}$. The prime example will be word counting functions and

context-free string rewriting systems. Hyperedge replacement systems [DKH97], the context-free systems of graph transformation, can be handled *mutatis mutandis*. The main result is PTIME computability of conditional means. For convenience, we extend the usage of the term locally algebraic as follows.

Definition 10 (Locally algebraic). *We call a q-matrix Q on* $\mathbf{S}$ locally algebraic *for an observable $f \in \mathbb{R}^{\mathbf{S}}$ if the set $\{Q^n f \mid n \in \mathbb{N}\}$, containing all multiple applications of the q-matrix Q to the observable f, is linearly dependent, i.e., if there exists a number $N \in \mathbb{N}$ such that the application $Q^N f$ of the N-th power is a linear combination $\sum_{i=0}^{N-1} \alpha_i Q^i f$ of lower powers of Q applied to f.*

Using local algebraicity of a q-matrix Q for an observable f, one can generate a *finite* ODE with one variable for each conditional mean $\mathsf{E}[Q^n f(X_t) \mid X_0 = x]$ (as detailed in the proof of Theorem 3); then, recent results from computable analysis [PG16] entail PTIME complexity.

Theorem 3 (PTIME complexity of conditional means). *Let Q be a q-matrix on $\mathbf{S}$, let $x \in \mathbf{S}$, let $f \colon \mathbf{S} \to \mathbb{R}$ be a function such that $f(x)$ is a PTIME computable real and $Q^N f = \sum_{i=0}^{N-1} \alpha_i Q^i f$ for some $N \in \mathbb{N}$ and PTIME computable coefficients α_i.*

The time evolution of the conditional mean $P_t f(x)$, i.e., the function $t \mapsto P_t f(x)$, is computable in polynomial time.

Proof. Consider the N-dimensional ODE with one variable E_n for each $n \in \{0, \dots, N-1\}$ with time derivative

$$\frac{\mathrm{d}}{\mathrm{d}t} E_n(t) = \begin{cases} E_{n+1}(t) & \text{if } n < N-1 \\ \sum_{i=0}^{N-1} \alpha_i E_i(t) & \text{if } n = N-1 \end{cases}$$

and initial condition $E_n(0) = Q^n f(x)$. Solving this ODE is in PTIME [PG16] (even over all of $\mathbb{R}_{\geq 0}$, using the "length" of the solution curve as implicit input). Finally, $E_i(t) = P_t Q^i f(x)$ is the unique solution.

Note that the linear ODE that we construct has a companion matrix as evolution operator, which allows one to use special techniques for matrix exponentiation [TR03, BCO+07].

Proposition 2 (Local algebraicity of context-free string rewriting). *Let $\mathcal{R}$ be a string rewriting system, let $w \in \Sigma^+$, let $m \in \mathbb{N}$. The q-matrix $Q_{\mathcal{R}}$ of the string rewriting system $\mathcal{R}$ is locally algebraic for the m-th power of w-occurrence counting $\sharp_w{}^m$ if $\mathcal{R} \subseteq \Sigma \times \Sigma^+$.*

Proof. For every product of word counting functions $\prod_{i=1}^{m} \sharp_{wi}$, applying the q-matrix $Q_{\mathcal{R}}$ to this product yields the observable $Q_{\mathcal{R}} \prod_{i=1}^{m} \sharp_{wi}$. Using previous work on graph transformation [DHHZS14, DHHZS15], restricted to acyclic, finite, edge labelled graphs that have a unique maximal path (with at least one edge), the observable $Q_{\mathcal{R}} \prod_{i=1}^{m} \sharp_{wi}$ is a linear combination $\sum_{j=1}^{k} \alpha_j \prod_{l=1}^{k_j} \sharp_{wj,l}$ of

word counting functions $\sharp_{wj,l}$ (with all $k_j \leq m$). Moreover, if $\mathcal{R}$ is context-free ($\mathcal{R} \subseteq \Sigma \times \Sigma^+$), we have $\sum_{l=1}^{k_j} |w_{j,l}| \leq \sum_{i=1}^{m} |w_i|$ for all $j \in \{1, \ldots, k\}$. Thus, we stay in a subspace that is spanned by a finite number of products of word counting functions.

Corollary 2. *For context-free string rewriting, conditional means and moments of word occurrence counts are computable in polynomial time.*

We conclude with a remark on lower bounds for the complexity.

Remark 1. The complexity of computing transient means, even for context-free string rewriting, is at least as hard as computing the exponential function. This becomes clear if we consider the rule $a \to aa$, and the observable of a-counts $\sharp_a$. Now, the time evolution of the $\sharp_a$-mean conditioned on the initial state to be a, i.e., the function $t \mapsto \mathsf{E}_a(\sharp_a(X_t))$, is exactly the exponential function e^t. Tight lower complexity bounds for the exponential function are a longstanding open problem [Ahr99].

7 Conclusion

The main result is computability of transient (conditional) means of Markov chains X_t "observed" by a function f, i.e., stochastic processes of the form $f(X_t)$. For this, we have described conditions under which a CTMC, specified by its q-matrix, induces a continuous-time *transformer* P_t that acts on observation functions. In analogy to predicate transformer semantics for programs, this could be called *observation transformer semantics* for CTMCs; formally, P_t is a strongly continuous semigroup on a suitable function space. Finally, motivated by important examples of context-free systems – be it the well-known class from Chomsky's hierarchy or the popular preferential attachment process (covered in previous work [DHHZS15]) – we have considered the special case of locally finite q-matrices. For this special case, we obtain a first complexity result, namely PTIME computability of transient conditional means.

The obvious next step is to implement our theoretical results since one cannot expect that the general algorithms of Weihrauch and Zhong [WZ07] perform well for every SCSG on a computable Banach space. For example, the Gauss-Jordan algorithm for infinite Matrices [Par12] should already be more practicable for inverting the operator $nI - \mathcal{Q}$ from Eq. (3) compared to the brute force approach used by Weihrauch and Zhong [WZ07]. Computability ensures existence of algorithms for computing transient means, but yields no guarantees of the efficiency of such algorithms.

Even if it should turn out that efficient algorithms are a pipe dream – after all, transient probabilities $p_{t,xy}$ are a special case of transient conditional means – we expect that already implementations that are slow but to arbitrary desired precision will be useful for gauging the quality of approximations of the "mean-field" of a Markov process, especially in the area of social networks [Gle13], but possibly also for chemical systems [SSG15]. Theoretically, they are a valid alternative to Monte Carlo simulation, or even preferable.

References

[ABHN11] Arendt, W., Batty, C.J.K., Hieber, M., Neubrander, F.: Vector-Valued Laplace Transforms and Cauchy Problems, vol. 96. Springer, Basel (2011)

[AFR11] Ackerman, N.L., Freer, C.E., Roy, D.M.: Noncomputable conditional distributions. In: Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, Ontario, Canada, pp. 107–116, 21–24 June 2011

[Ahr99] Ahrendt, T.: Fast computations of the exponential function. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 302–312. Springer, Heidelberg (1999). doi:10.1007/3-540-49116-3_28

[And91] Anderson, W.J.: Continuous-Time Markov Chains. Springer, New York (1991)

[Aus55] Austin, D.G.: On the existence of the derivative of Markoff transition probability functions. Proc. Natl. Acad. Sci. USA **41**(4), 224–226 (1955)

[BCO+07] Bostan, A., Chyzak, F., Ollivier, F., Salvy, B., Schost, É, Sedoglavic, A.: Fast computation of power series solutions of systems of differential equations. In: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, Philadelphia, PA, USA, pp. 1012–1021. Society for Industrial and Applied Mathematics (2007)

[BHHK03] Baier, C., Haverkort, B., Hermanns, H., Katoen, J.-P.: Model-checking algorithms for continuous-time Markov chains. IEEE Trans. Softw. Eng. **29**(6), 524–541 (2003)

[DFF+10] Danos, V., Feret, J., Fontana, W., Harmer, R., Krivine, J.: Abstracting the differential semantics of rule-based models: exact and automated model reduction. In: Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, Edinburgh, United Kingdom, pp. 362–381, 11–14 July 2010

[DHHZS14] Danos, V., Heindel, T., Honorato-Zimmer, R., Stucki, S.: Approximations for stochastic graph rewriting. In: Merz, S., Pang, J. (eds.) ICFEM 2014. LNCS, vol. 8829, pp. 1–10. Springer, Cham (2014). doi:10.1007/978-3-319-11737-9_1

[DHHZS15] Danos, V., Heindel, T., Honorato-Zimmer, R., Stucki, S.: Moment semantics for reversible rule-based systems. In: Krivine, J., Stefani, J.-B. (eds.) RC 2015. LNCS, vol. 9138, pp. 3–26. Springer, Cham (2015). doi:10.1007/978-3-319-20860-2_1

[DKH97] Drewes, F., Kreowski, H.-J., Habel, A.: Hyperedge: replacement, graph grammars. In: Rozenberg, G. (ed.) Handbook of Graph Grammars and Computing by Graph Transformation, pp. 95–162. World Scientific, Singapore (1997)

[Ein52] Einar, H.: A note on Cauchy's problem. Annales de la Société Polonaise de Mathématique **25**, 56–68 (1952)

[EN00] Engel, K.-J., Nagel, R.: One-Parameter Semigroups for Linear Evolution Equations. Springer, New York (2000)

[Gle13] Gleeson, J.P.: Binary-state dynamics on complex networks: pair approximation and beyond. Phys. Rev. X **3**, 021004 (2013)

[GM84] Gross, D., Miller, D.R.: The randomization technique as a modeling tool and solution procedure for transient Markov processes. Oper. Res. **32**(2), 343–361 (1984)

[HLM06] Heckel, R., Lajios, G., Menge, S.: Stochastic graph transformation systems. Fundamenta Informaticae **74**(1), 63–84 (2006)

[Kol51] Andrey Nikolaevich Kolmogorov: On the differentiability of the transition probabilities in stationary Markov processes with a denumberable number of states. Moskovskogo Gosudarstvennogo Universiteta Učenye Zapiski Matematika **148**, 53–59 (1951)

[Koz83] Kozen, D.: A probabilistic PDL. In: Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing, STOC 1983, pp. 291–297. ACM, New York (1983)

[Nor98] Norris, J.R.: Markov Chains. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, Cambridge (1998)

[Par12] Paraskevopoulos, A.G.: The infinite Gauss-Jordan elimination on row-finite $\omega \times \omega$ matrices. arXiv preprint math (2012)

[PG16] Pouly, A., Graça, D.S.: Computational complexity of solving polynomial differential equations over unbounded domains. Theor. Comput. Sci. **626**, 67–82 (2016)

[Reu57] Reuter, G.E.H.: Denumerable Markov processes and the associated contraction semigroups on l. Acta Mathematica **97**(1), 1–46 (1957)

[RR72] Reuter, G.E.H., Riley, P.W.: The Feller property for Markov semigroups on a countable state space. J. Lond. Math. Soc. **s2–5**(2), 267–275 (1972)

[Spi12] Spieksma, F.M.: Kolmogorov forward equation and explosiveness in countable state Markov processes. Ann. Oper. Res. **241**, 3–22 (2012)

[Spi15] Spieksma, F.M.: Countable state Markov processes: non-explosiveness and moment function. Probab. Eng. Inf. Sci. **29**, 623–637 (2015)

[Spi16] Spieksma, F.M.: Personal communication, October 2016

[SSG15] David Schnoerr, Guido Sanguinetti, Ramon Grima: Comparison of different moment-closure approximations for stochastic chemical kinetics. J. Chem. Phys. **143**(18) (2015)

[TR03] Rajae Ben Taher and Mustapha Rachidi: On the matrix powers and exponential by the r-generalized fibonacci sequences methods: the companion matrix case. Linear Algebra Appl. **370**, 341–353 (2003)

[VMS93] Van Moorsel, A.P., Sanders, W.H.: Adaptive uniformization: technical details. Technical report, Department of Computer Science and Department of Electrical Engineering, University of Twente (1993)

[VMS94] Van Moorsel, A.P., Sanders, W.H.: Adaptive uniformization. Commun. Stat. Stoch. Models **10**, 619–647 (1994)

[WZ07] Weihrauch, K., Zhong, N.: Computable analysis of the abstract Cauchy problem in a Banach space and its applications I. Math. Logic Q. **53**(4–5), 511–531 (2007)

Pointless Learning

Florence Clerc[1], Vincent Danos[2,4], Fredrik Dahlqvist[3], and Ilias Garnier[4(✉)]

[1] McGill University, Montreal, Canada
[2] ENS Paris/CNRS, Paris, France
[3] UCL, London, UK
[4] University of Edinburgh, Edinburgh, Scotland
ilias.gar@gmail.com

Abstract. Bayesian inversion is at the heart of probabilistic programming and more generally machine learning. Understanding inversion is made difficult by the pointful (kernel-centric) point of view usually taken in the literature. We develop a pointless (kernel-free) approach to inversion. While doing so, we revisit some foundational objects of probability theory, unravel their category-theoretical underpinnings and show how pointless Bayesian inversion sits naturally at the centre of this construction.

1 Introduction

The soaring success of Bayesian machine learning has yet to be matched with a proper foundational understanding of the techniques at play. These statistical models are fundamentally programs that manipulate probability distributions. Therefore, the semantics of programming languages can and should inform the semantics of machine learning. This point of view, upheld by the proponents of *probabilistic programming*, has given rise to a growing body of work on matters ranging from the computability of disintegrations [1] to operational and denotational semantics of probabilistic programming languages [12]. These past approaches have all relied on a pointful, *kernel-centric* view of the key operation in Bayesian learning, namely *Bayesian inversion*. In this paper, we show that a *pointless*, operator-based approach to Bayesian inversion is both more general, simpler and offers a more structured view of Bayesian machine learning.

Let us recall the underpinnings of Bayesian inversion in the finite case. Bayesian statistical inference is a method for updating subjective probabilities on an unknown random process as observations are collected. In a finite setting, this update mechanism is captured by Bayes' law:

$$P(d) \cdot P(h \mid d) = P(d \mid h) \cdot P(h) \tag{1}$$

F. Clerc—Research supported by NSERC, Canada.
F. Dahlqvist—Research partially supported by the ERC project *ProFoundNet - Probabilistic Foundations for Networks* (grant number 679127).
I. Garnier—Research supported by the ERC project *RULE* (grant number 320823).

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 355–369, 2017.
DOI: 10.1007/978-3-662-54458-7_21

On the right-hand side, the *likelihood* $P(d \mid h)$ encodes a parameter-dependent probability over data, weighted by the *prior* $P(h)$ which corresponds to our current belief on which parameters best fit the law underlying the unknown random process. The left-hand side of Eq. 1 involves the *marginal likelihood* $P(d)$, which is the probability of observing the data d under the current subjective probability, and the *posterior* $P(h \mid d)$ which tells us how well the occurrence of d is explained by the parameter h. More operationally, the posterior tells us how we should revise our prior as a function of the observed data d. In a typical Bayesian setup, the prior and likelihood are given and the marginal likelihood can be computed from the two first ingredients. The only unknown is the *posterior* $P(h \mid d)$. Equation 1 allows one to compute the posterior from the two first ingredients–whenever $P(d) > 0$! This formulation emphasises the fundamental symmetry between likelihood and posterior, and hopefully makes clear why the process of computing the posterior is called *Bayesian inversion*. The key observation is that both the likelihood and posterior can be seen as matrices, and Eq. 1 encodes nothing more than a relation of adjunction between these matrices seen as (finite-dimensional) operators. This simple change of point of view, where one thinks no longer directly in terms of kernels (which transform probability measures forward), but in terms of their semantics as operators (which transform real-valued obervables backward) generalises well and gives us a much more comprehensive account of Bayesian learning as adjunction. If one thinks of observables as extended predicates, this change of point of view is nothing but a predicate transformer semantics of kernels: a well-established idea planted in the domain of probabilistic semantics by Kozen in the 80s [10]. The object of this paper is to develop in this setting a *pointless* approach to Bayesian inversion.

Our contributions are as follows. In Sect. 3, we recall how Bayesian inversion is formulated using the language of kernels, following the seminal work of [5] and our own preliminary elaboration of the ideas developed in the current paper [6]. The adequate setting is a category of *typed* kernels, i.e. measure-preserving kernels between probability spaces. We observe that Bayesian inversion fits somewhat awkwardly in this pointful setting. Drawing from domain-theoretic ideas [11], we develop in Sect. 4 a categorical theory of ordered Banach cones, including an adjunction theorem for L_p^+/L_q^+ cones taken from Reference [3]. In Sect. 5, we define a functorial operator interpretation of kernels in the category of Banach cones and prove that pointful Bayesian inversion corresponds through this functorial bridge to adjunction, expanding our recent result [6] to arbitrary L_p^+/L_q^+ cones. Unlike the pointful case, the pointless, adjunction-based approach works with arbitrary measurable spaces. Finally, in Sect. 6 we extract from the pointful and pointless approaches what we consider to be the essence of Bayesian inversion: a correspondence between *couplings* and linear operators. In this new light, adjunction (and therefore Bayesian inversion) is nothing more than a permutation of coordinates. We conclude with a sketch of some directions for future research where one could most profit of the superior agility and extension of the pointless approach.

Note that a long version of this article, containing all proofs, is available [4].

2 Preliminaries

We refer the reader to e.g. [2] for the concepts of measure theory and functional analysis used in this paper. For convenience, some basic definitions are recalled here.

A *measurable space* (X, Σ) is given by a set X together with a σ-algebra of subsets of X denoted by Σ. Where unambiguous, we will omit the σ-algebra and denote a measurable space by its underlying set. We will also consider the measurable spaces generated from *Polish* (completely metrisable and separable) topological spaces, called *standard Borel spaces* [9]. A measurable function $f : (X, \Sigma) \to (Y, \Lambda)$ is a function $f : X \to Y$ such that for all $B \in \Lambda$, $f^{-1}(B) \in \Sigma$. The category of measurable spaces and measurable functions will be denoted by **Mes**. For B a measurable set, we denote by $\mathbb{1}_B$ the indicator function of that set. A *finite measure* μ over a measurable space (X, Σ) is a σ-additive function $\mu : \Sigma \to [0, \infty)$ that verifies $\mu(X) < \infty$. Whenever $\mu(X) = 1$, μ is a *probability measure*. A pair (X, μ) with X a measurable space and μ a probability measure on X is called a *probability space*. A measurable set B will be qualified of *μ-null* if $\mu(B) = 0$.

The Giry endofunctor, denoted by $\mathsf{G} : \mathbf{Mes} \to \mathbf{Mes}$, maps each measurable space X to the space $\mathsf{G}(X)$ of probability measures over X. The measurable structure of $\mathsf{G}(X)$ is given by the initial σ-algebra for the family $\{ev_B\}_B$ of evaluation maps $ev_B(\mu) = \mu(B)$, where B ranges over measurable sets in X. The action of G on arrows is given by the pushforward (or image measure): for $f : X \to Y$ measurable, we have $\mathsf{G}(f) : \mathsf{G}(X) \to \mathsf{G}(Y)$ given by $\mathsf{G}(f)(\mu) = \mu \circ f^{-1}$. This functor admits the familiar monad structure (G, m, δ) where $m : \mathsf{G}^2 \Rightarrow \mathsf{G}$ and $\delta : Id \Rightarrow \mathsf{G}$ are natural transformations with components at X defined by $m_X(P)(B) = \int_{\mathsf{G}(X)} ev_B \, dP$ and $\delta_X(x)(B) = \delta_x(B)$. It is well-known that when restricted to standard Borel spaces, the Giry functor admits the same monad structure. See [7] for more details on this construction. The Kleisli category of the Giry monad, corresponding to Lawvere's category of probabilistic maps, will be denoted by $\mathcal{K}\ell$. The objects of $\mathcal{K}\ell$ correspond to those of **Mes** and arrows from X to Y correspond to so-called *kernels* $f : X \to \mathsf{G}(Y)$. Kleisli arrows will be denoted by $f : X \nrightarrow Y$. For $f : X \nrightarrow Y, g : Y \nrightarrow Z$, the Kleisli composition is defined as usual by $g \circ' f = m_Z \circ \mathsf{G}(g) \circ f$. We distinguish deterministic Kleisli maps as those that can be factored as a measurable function followed by δ and denote these arrows $f : X \nrightarrow_\delta Y$. We write 1 for the one element measurable space (which is the terminal object in **Mes**). Clearly the Homset $\mathcal{K}\ell(1, Y)$ is in bijection with the set of probabilities over Y. This justifies the following slight abuse of notation: if $\mu \in \mathsf{G}(X)$ is a probability and $f : X \nrightarrow Y$ is a kernel, the pushforward of μ through f will be denoted $f \circ' \mu$. Observe that for $f : X \to Y$ an usual measurable map, $\mathsf{G}(f)(\mu) = (\delta_Y \circ f) \circ' \mu$, so the pushforward through a kernel extends the earlier definition.

Consider the full subcategory of $\mathcal{K}\ell$ restricted to finite spaces. In that setting, any kernel $f : X \nrightarrow Y$ can be presented as a positive, real-valued matrix that we denote $T(f) = \{f(x)(y)\}_{x,y}$ with X rows, Y columns and where all rows sum to 1 (aka a stochastic matrix). Matrix multiplication corresponds to Kleisli

composition: taking f, g as above, one has $T(g \circ' f) = T(f)T(g)$ (hence, this representation of kernels as matrices is *contravariant*). Such matrices act on vectors of dimension Y (*observables on* Y) and map them to observables on X: for $v \in \mathbb{R}^Y$, $T(f)v$ corresponds to the expectation of v according to f. This is the basis for the "operator interpretation" of kernels which we will extend to **Mes** below.

3 Bayesian Inversion in a Category of Typed Kernels

We introduce the category **Krn** of typed kernels and recall the statement of Bayesian inversion in this setting.

3.1 Definition of Krn

Our starting point is the under category $1 \downarrow \mathcal{K}\ell$, where 1 is the one-element measurable space. Objects of $1 \downarrow \mathcal{K}\ell$ are Kleisli arrows $\mu : 1 \nrightarrow X$, i.e. probability spaces (X, μ) with $\mu \in \mathsf{G}(X)$; while typed kernels from (X, μ) to (Y, ν) are Kleisli arrows $f : X \nrightarrow Y$ such that $f \circ' \mu = \nu$. We will call these arrows "kernels" for short. For a deterministic map $f_\delta : X \nrightarrow_\delta Y$ (factoring as $f_\delta = \delta_Y \circ f$), the constraint boils down to $\nu = \mathsf{G}(f)(\mu)$. In other words, the subcategory of $1 \downarrow \mathcal{K}\ell$ consisting of deterministic maps is isomorphic to the usual category of probability spaces and measure-preserving maps. We define **Krn** to be the subcategory of $1 \downarrow \mathcal{K}\ell$ restricted to standard Borel spaces.

3.2 Bayesian Inversion in the Finite Subcategory of Krn

We translate the presentation of Bayesian inversion of Sect. 1 in the language of **Krn**. We are given finite spaces of data D and parameters H and it is assumed that there exists an unknown probability on D, called the "truth" and denoted τ in the following, that we wish to learn. The likelihood corresponds to a $\mathcal{K}\ell$ arrow $f : H \nrightarrow D$, The prior is a probability $\mu \in \mathsf{G}(H)$ while the marginal likelihood $\nu \in \mathsf{G}(D)$ is obtained as $\nu = f \circ' \mu$. Thus the entire situation is captured by a **Krn** arrow $f : (H, \mu) \nrightarrow (D, \nu)$. *If* our prior was perfect, we would have $\nu = \tau$ but of course (by assumption) this is not the case! The only access we have to the truth is through an infinite, independent family $\{d_n\}_{n \in \mathbb{N}}$ of random elements in D each distributed according to τ. The Bayesian update is the process of using this sequence of data (sometimes called *evidence*) to iteratively revise our prior. In this language, Bayes's law reads as follows:

$$\nu(d) \cdot f^\dagger(d)(h) = f(h)(d) \cdot \mu(h) \tag{2}$$

where $f^\dagger : (D, \nu) \nrightarrow (H, \mu)$ denotes the sought posterior map, to be computed in function of μ and f. Observe that both the left and right hand side of Eq. 2 define the same *joint probability* $\gamma \in \mathsf{G}(H \times D)$ given by $\gamma(h, d) = f(h)(d) \cdot \mu(h) = \nu(d) \cdot f^\dagger(d)(h)$. Denoting π_H, π_D the left and right projections from $H \times D$, one easily verifies that $\mathsf{G}(\pi_H) = \mu$ and $\mathsf{G}(\pi_D) = \nu$. In other terms, γ is a *coupling* of μ and ν. We draw the attention of the reader to the following points.

– As hinted before, $f^\dagger(d)$ is uniquely defined only when $\nu(d) > 0$. Conversely, $f^\dagger$ does not depend on f on μ-null sets. These hurdles will be circumvented by considering equivalence classes of kernels up to null sets. This is the object of Sect. 3.3.
– Section 2 introduces a correspondence between (finite) kernels and Markov or stochastic matrices. This raises the following question: what is Bayesian inversion seen through that lens? The answer is *adjunction*. As we show in Sect. 5, this *pointless* point of view generalises to arbitrary measurable spaces and is better behaved than the pointful one.

We now proceed to the generalisation of this situation to the case of standard Borel spaces, i.e. to that of **Krn**.

3.3 Bayesian Inversion in Krn

Bayesian inversion in **Krn** relies crucially on the construction of an (almost sure) bijection between the **Krn** Homset $\mathbf{Krn}(X, \mu; Y, \nu)$ and the set of couplings $\mathbf{\Gamma}(X, \mu; Y, \nu)$ of μ and ν (to be defined next).

Couplings and kernels. To any pair of objects $(X, \mu)(Y, \nu)$, one can associate the space of *couplings* of μ and ν, i.e. the set of all probabilities $\gamma \in \mathsf{G}(X \times Y)$ such that $\mathsf{G}(\pi_X)(\gamma) = \mu$ and $\mathsf{G}(\pi_Y)(\gamma) = \nu$. We denote this set of couplings $\mathbf{\Gamma}(X, \mu; Y, \nu)$. It is a standard Borel space, as the set of couplings of two measures is a closed convex subset in $\mathsf{G}(X \times Y)$ for any choice of a Polish topology for X, Y. In order to construct a mapping from couplings to **Krn** arrows, we will need the *disintegration theorem*, which requires us to introduce some terminology. In the following, we denote $N(f, f') = \{x \mid f(x) \neq f'(x)\}$.

Lemma 1. *For all $f, f' : (X, \mu) \nrightarrow (Y, \nu)$, $N(f, f')$ is measurable.*

Note that in more general measurable spaces, $N(f, f')$ is not necessarily a measurable set, as those spaces are not always countably generated. We can now introduce the disintegration theorem.

Theorem 1 (Disintegration ([8], Theorem 5.4)). *For all deterministic* **Krn** *arrow $f : (X, \mu) \nrightarrow_\delta (Y, \nu)$, there exists $f^\dagger : (Y, \nu) \nrightarrow (X, \mu)$ such that $f \circ' f^\dagger = id_{(Y,\nu)}$ and such that for all $h : (Y, \nu) \nrightarrow (X, \mu)$ verifying $f \circ' h = id_{(Y,\nu)}$, $\nu(N(f^\dagger, h)) = 0$. In short, we say that $f^\dagger$ is the ν-almost surely unique kernel verifying $f \circ' f^\dagger = id_{(Y,\nu)}$.*

Disintegrations correspond to *regular conditional probabilities* (see e.g. [8]). The deterministic map $f : X \to Y$ along which the disintegration of μ is computed acts through its fibers as a parameterised family of subsets on each of which μ is conditioned, resulting in a measurable family of conditional probabilities parameterised by Y. Note that the characteristic property of disintegrations can be equivalently stated as the fact that $f^\dagger(y)$ is ν-almost surely supported by $f^{-1}(y)$.

Example 1. In the finite case, disintegration is simply the formula for conditional probabilities. Given X, Y finite and $f : (X, \mu) \nrightarrow_\delta (Y, \nu)$, for $y \in Y$ s.t. $\nu(y) = \mu(f^{-1}(y)) > 0$, it holds that $f^\dagger(y)(x) = \frac{\mu(x)}{\nu(y)}$. However, when $\nu(y) = 0$, the disintegration theorem does not constrain the value of $f^\dagger(y)$.

Disintegration establishes a bijective (up to null sets) correspondence between couplings and kernels. Let us make this formal.

Definition 1. *For fixed $(X, \mu)(Y, \nu)$, we define on $\mathbf{Krn}(X, \mu; Y, \nu)$ $\sim$ as the smallest equivalence relation such that $f \sim f'$ if $\mu(N(f, f')) = 0$. We denote $\mathbf{Krn}(X, \mu; Y, \nu)/\mu$ the set of $\sim$-equivalence classes of $\mathbf{Krn}(X, \mu; Y, \nu)$.*

Any **Krn** arrow $f : (X, \mu) \nrightarrow (Y, \nu)$ induces a measure on $X \times Y$, defined on measurable rectangles $B_X \times B_Y$ as:

$$I^{Y,\nu}_{X,\mu}(f)(B_X \times B_Y) = \int_{x \in B_X} f(x)(B_Y)\, d\mu. \tag{3}$$

Lemma 2. *$I^{Y,\nu}_{X,\mu}$ is a* **Set** *injection from $\mathbf{Krn}(X, \mu; Y, \nu)/\mu$ to $\mathbf{\Gamma}(X, \mu; Y, \nu)$.*

The second part of the bijection between couplings and quotiented **Krn** arrows relies crucially on disintegration.

Lemma 3. *There is a* **Set** *injection $D^{Y,\nu}_{X,\mu} : \mathbf{\Gamma}(X, \mu; Y, \nu) \to \mathbf{Krn}(X, \mu; Y, \nu)/\mu$. Moreover, $D^{Y,\nu}_{X,\mu}$ and $I^{Y,\nu}_{X,\mu}$ are inverse of one another.*

Bayesian inversion in **Krn**. Bayesian inversion corresponds to the composition of the bijections we just defined with the pushforward along the permutation map $\sigma : X \times Y \to Y \times X$.

Theorem 2 (Bayesian inversion). *Let $-^\dagger$ be defined as $f^\dagger = D^{X,\mu}_{Y,\mu} \circ \mathsf{G}(\sigma) \circ I^{Y,\nu}_{X,\mu}$. The map $-^\dagger : \mathbf{Krn}(X, \mu; Y, \nu)/\mu \to \mathbf{Krn}(Y, \nu; X, \mu)/\nu$ is a bijection.*

This section would be incomplete if we didn't address *learning* in its relation to Bayesian inversion. It is known that in good cases,[1] Bayesian inversion will make the sequence of marginal likelihoods converge to the truth in some appropriate topology. However, issues of convergence are not the subject of this paper and will not be discussed further.

3.4 Pointfulness Is Harmful

Let us take a critical look at the approach to Bayesian inversion developed so far. The fact that $-^\dagger$ is by construction $\sim$-invariant and yields $\sim$-equivalence classes of **Krn** arrows suggests that $-^\dagger$ would be better typed on a hypothetical quotient of **Krn** by $\sim$. This mismatch between the behaviour of $-^\dagger$ and its actual type already arises in the finite case where Bayes' rule yields kernels

[1] E.g. H, D finite and μ putting strictly positive measure on $f^{-1}(\tau)$.

only defined up to a null set (see discussion after Eq. 2), and is an inevitable consequence of the pointful point of view: kernels should respect the measures endogenous to their domain. Constructing the quotient of **Krn** w.r.t. $\sim$ would require proving that this equivalence relation is compatible with the composition of **Krn**. However, carrying out this approach successfully seems non-trivial:[2] our past attempts are riddled with obstructions stemming from accumulation of negligible sets–the very technical hurdles that make the theory of disintegration of measures so unintuitive in the first place, while moreover relying on standard Borel assumptions.

This improper typing obscures the categorical structure of Bayesian inversion. In the next sections, we leave the inhospitable world of kernels and relocate the theory of Bayesian inversion in a category of Banach cones and linear maps where these problems vanish, and the structure we seek for becomes manifest.

4 Banach Cones

Following [3,11], we introduce a category of Banach cones and ω-continuous linear maps, with the intent of interpreting Markov kernels as linear operators between well-chosen function spaces. In the subcategory corresponding to these function spaces, we develop a powerful adjunction theorem that will be used in Sect. 5 to implement pointless Bayesian inversion.

4.1 The Category Ban

A *Banach cone*, informally, corresponds to a normed convex cone of a Banach space which is ω-complete with respect to a particular order. Let us introduce these cones progressively.

Definition 2. *A normed, convex cone* $(C, +, \cdot, 0, \|\cdot\|_C)$ *of a normed vector space* $(V, +, \cdot, 0, \|\cdot\|_V)$ *is a subset* $C \subseteq V$ *that is closed under addition, convex combinations and multiplications by non-negative scalars, endowed with the restriction of the ambient norm, which must be monotone w.r.t. the partial order* $u \leq_C v \Leftrightarrow \exists w \in C. u + w = v$.

We require our Banach cones to be ω-*complete* with respect to this order, and to be subsets of Banach spaces.

Definition 3 (Banach cones). *A normed convex cone* C *is* ω-complete *if for all chain (i.e.* $\leq_C$*-increasing countable family)* $\{u_n\}_{n\in\mathbb{N}}$ *of bounded norm, the least upper bound* $\bigvee_n u_n$ *exists and* $\|\bigvee_n u_n\|_C = \bigvee_n \|u_n\|_C$. *A* Banach cone *is an* ω*-complete normed cone of a Banach space.*

Norm convergence and order convergence are related by the following result.

[2] Without additional assumptions the quotient is not compatible with pre-composition, differently to what we mistakenly stated in ([6], Lemma 3).

Lemma 4 ([3], Lemma 2.12). *Let* $\{u_n\}_{n\in\mathbb{N}}$ *be a chain of bounded norm in a Banach cone. Then* $\lim_{i\to\infty} \|\bigvee_n u_n - u_i\| = 0$.

A prime example of Banach cones is given by the *positive cones* associated to classical L_p spaces of real-valued functions (see e.g. [2] for a definition of those spaces). In details: for (X, μ) a measure space with μ finite and $p \in [1, \infty]$, the set of elements $f \in L_p(X, \mu)$ which are non-negative μ-a.e. is closed under addition, multiplication by non-negative scalars and under linear combinations with non-negative coefficients. Equipped with the restriction of the norm of $L_p(X, \mu)$, this subset forms a normed convex cone that we denote $L_p^+(X, \mu)$. The partial order associated to these L_p^+ cones can be defined explicitly: for $f, g \in L_p^+(X, \mu)$, we write that $f \leq g$ if $f(x) \leq g(x)$ μ-a.e. One easily checks that this coincides with the definitional partial order.

Proposition 1 (ω-completeness of L_p^+ cones, [3]). *For all* X *measurable,* $\mu \in \mathsf{G}(X)$ *and* $p \in [1, \infty]$, $L_p^+(X, \mu)$ *is a Banach cone.*

This result is a direct consequence of the definition of suprema in $L_p^+(X, \mu)$. We are going to construct a category of all Banach cones and we thus have to specify what a morphism between such cones is. We consider only linear maps which are Scott-continuous, which in this case[3] boils down to commuting with supremas of increasing chains.

Definition 4. *Let* C, C' *be Banach cones and* $A : C \to C'$ *be a linear map.* A *is* ω*-continuous if for every chain* $\{f_n\}_{n\in\mathbb{N}}$ *such that* $\bigvee_n f_n$ *exists,* $A(\bigvee_n f_n) = \bigvee_n A(f_n)$.

The following example should help make ω-continuity less mysterious. Observe that for $Y = 1$ (the singleton set), all Banach cones $L_p^+(Y, \mu)$ (for μ nonzero, otherwise $L_p^+(Y, \mu) \cong \{0\}$) are isomorphic to $\mathbb{R}_{\geq 0}$ – therefore, $\mathbb{R}_{\geq 0}$ is a *bona fide* Banach cone.

Example 2. There exists a familiar linear map from $L_p^+(X, \mu)$ to $\mathbb{R}_{\geq 0}$, namely the Lebesgue integral $\int : L_p^+(X, \mu) \to \mathbb{R}_{\geq 0}$, taking $u \in L_p^+(X, \mu)$ to $\int_X u \, d\mu$. In this case, ω-continuity of the integral is simply the monotone convergence theorem!

Unless stated otherwise, all maps in the remainder of this section are ω-continuous. The property of ω-continuity is closed under composition and the identity function is trivially ω-continuous. This takes us to the following definition.

Definition 5 (Categories of Banach cones and of L_p^+ cones). *The category* **Ban** *has Banach cones as objects and* ω*-continuous linear maps as morphisms. We distinguish the full subcategory* **L** *having as objects all* L_p^+*-spaces (ranging over all* $p \in [1, \infty]$*). Further,* **L** *admits a family of full subcategories* $\{\mathbf{Lp}\}_{\mathbf{p}\in[1,\infty]}$*, each having as objects* L_p^+ *spaces (for fixed* p*).*

[3] These cones have the "countable sup property"[2]. Therefore, all directed sets admit a countable subset having the same least upper bound, and we can restrict our attention to chains.

Ban is itself a full subcategory of the category $\omega\mathbf{CC}$ of ω-complete normed cones and ω-continuous maps, as defined in [3]. Let us denote by $\mathbf{Ban}(C, C')$ the set of ω-continuous linear maps from C to C'. Denoting $\|\cdot\|_C$ the norm of C, we recall that the operator norm of a linear map $A : C \to C'$ is given by $\|A\|_{op} = \inf\{K \geq 0 \mid \forall\, u \in C, \|Au\|_{C'} \leq K \|u\|_C\}$. A partial order on $\mathbf{Ban}(C, C')$ is given by $A \leq B$ *iff* for all $u \in C$, $A(u) \leq_{C'} B(u)$. Selinger proved in [11] that ω-continuous linear maps between ω-complete cones have automatically bounded norm (i.e. they are continuous in the usual sense), therefore we can and will abstain from asking continuity explicitly. The following result is a cone-theoretic counterpart to the well-known fact that the vector space of bounded linear operators between two Banach spaces forms a Banach space for the operator norm.

Proposition 2. *For all Banach cones C, C', the cone of ω-continuous linear maps $\mathbf{Ban}(C, C')$ is a Banach cone for the operator norm and the pointwise order.*

4.2 Duality in Banach Cones

We use a powerful Banach cone duality result initially proved in the supplementary material to [3]. We say that a pair (p, q) with $p, q \in [1, \infty]$ is *Hölder conjugate* if $\frac{1}{p} + \frac{1}{q} = 1$. For any Banach cone C, its dual C^* is by definition the Banach cone of ω-continuous linear functionals, i.e. the cone $C^* = \mathbf{Ban}(C, \mathbb{R}_{\geq 0})$. This operation defines a contravariant endofunctor $-^* : \mathbf{Ban} \to \mathbf{Ban}^{op}$ mapping each cone C to C^* and each map of cone $A : C \to C'$ to the map $A^* : C'^* \to C^*$ defined by $A^*(\varphi) = \varphi \circ A$, for $\varphi \in C'^*$. For Hölder conjugate (p, q), we have the following extension to the classical isomorphism of L_p^+ spaces.

Theorem 3 (L_p^+ cone duality [3]). *There is a Banach cone isomorphism $\varepsilon_p : L_p^+(X, \mu) \cong L_q^{+,*}(X, \mu)$.*

We won't reproduce the proof of this theorem here, which can be found in the supplementary material to [3]. Suffice it to say it is a Riesz duality type argument which relies entirely on the Radon-Nikodym theorem. Note that Theorem 3 implies in particular that $L_\infty^{+,*}(X, \mu) \cong L_1^+(X, \mu)$, which classically fails in the usual setting of L_p Banach spaces. It is instructive to study how ω-continuity wards off a classical counter-example to duality in the general Banach case.

Example 3 (Taken from [11]). Let μ be a probability measure on $\mathbb{N}$ with full support. We consider the cone $\ell_\infty^+ = L_\infty^+(\mathbb{N}, \mu)$ of bounded sequences of real numbers. Let $\mathcal{U}$ be a non-principal ultrafilter on $\mathbb{N}$ (i.e. an ultrafilter on the partial order of subsets of $\mathbb{N}$ without a least element). We define the function $\lim_{\mathcal{U}} : \ell_\infty^+ \to \mathbb{R}$ as $\lim_{\mathcal{U}}(\{x_n\}_{n\in\mathbb{N}}) = \sup\{y \mid \{n \mid x_n \geq y\} \in \mathcal{U}\}$. This function is linear and bounded. However, consider the chain $\left\{u^k \in \ell_\infty^+\right\}_{k\in\mathbb{N}}$ with $u_n^k = 1$ for all $n \leq k$ and $u_n^k = 0$ for all $n > k$. The supremum of this chain is the constant 1 sequence. On the other hand, we have $\lim_{\mathcal{U}}(u^k) = 0$ for all k, whereas $\lim_{\mathcal{U}}(\bigvee_k u^k) = 1$. Therefore, $\lim_{\mathcal{U}}(u^k)$ is not ω-continuous–i.e., $\lim_{\mathcal{U}} \notin \ell_\infty^{+,*}$.

It is useful to have a concrete representation of the isomorphism stated in Theorem 3. This theorem implies that for all $u \in L_p^+(X,\mu)$, there exists a unique ω-continuous linear functional $\varepsilon(u) \in L_q^{+,*}(X,\mu)$–which must therefore correspond to $\varepsilon(u)(v) = \int_X uv\, d\mu$. The pairing between L_p^+ and L_q^+ cones that we introduce below corresponds to the evaluation of such a functional against some argument.

Definition 6 (Pairing). *For Hölder conjugate (p,q), the* pairing *is the map $\langle \cdot,\cdot \rangle_X : L_p^+(X,\mu) \times L_q^+(X,\mu) \to \mathbb{R}_{\geq 0}$ defined by $\langle u, u' \rangle = \int uu'\, d\mu$.*

The pairing is bilinear, continuous and ω-continuous in each argument (consequences of the corresponding properties of the Lebesgue integral). We can now state the adjunction theorem.

4.3 Adjunctions Between Conjugate L_p^+ Cones

It is instructive to look at Theorem 3 under a slightly more general light. Observe that $L_p^+(X,\mu)$ is isomorphic to $\mathbf{Ban}(\mathbb{R}_{\geq 0}, L_p^+(X,\mu))$: indeed, any map A in this function space is entirely constrained by linearity by its value at 1. Therefore, Theorem 3 really states a Banach cone isomorphism between $\mathbf{Ban}(\mathbb{R}_{\geq 0}, L_p^+(X,\mu))$ and $\mathbf{Ban}(L_q^+(X,\mu), \mathbb{R}_{\geq 0})$. This isomorphism generalises to the case where $\mathbb{R}_{\geq 0}$ is replaced by an arbitrary conjugate pair of cones $L_p^+(Y,\mu), L_q^+(Y,\nu)$ (i.e. s.t. (p,q) are Hölder conjugate).

Theorem 4 (L_p^+/L_q^+ adjunction). *For (p,q) Hölder conjugate and for all $A : L_p^+(X,\mu) \to L_p^+(Y,\nu)$, $A^* : L_q^+(Y,\nu) \to L_q^+(X,\mu)$ is unique such that*

$$\forall\, u \in L_p^+(X,\mu), v \in L_q^+(Y,\nu),\ \langle v, A(u) \rangle_Y = \langle A^*(v), u \rangle_X. \tag{4}$$

The essence of the previous theorem is neatly captured as follows.

Corollary 1. *For all Hölder conjugate (p,q), the duality functor $-^* : \mathbf{Ban} \to \mathbf{Ban}^{op}$ restricts to an equivalence of categories $-^* : \mathbf{Lp} \to \mathbf{Lq}^{op}$.*

Figure 1 recapitulates the categories of Banach cones mentioned in this section along their relationships.

5 Pointless Bayesian Inversion

Krn arrows can be represented as linear maps between function spaces. This bridge allows one to manipulate Markov kernels both from the measure-theoretic side and from the functional-analytic side. Concretely, this linear interpretation of kernels is presented as a family of functors from **Krn** to **L**, the subcategory of **Ban** restricted to L_p^+ cones and ω-continuous linear maps. We show that pointful Bayesian inversion, whenever it is defined, coincides with adjunction.

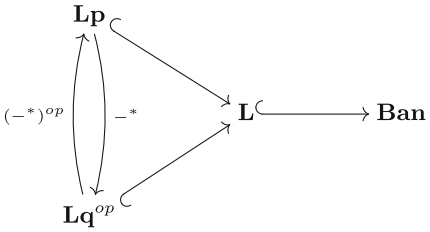


Fig. 1. Categories of cones

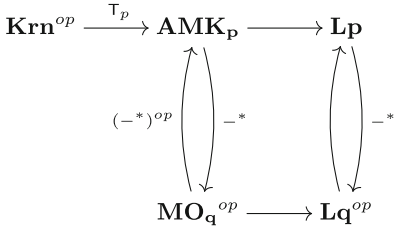


Fig. 2. Kernels, AMKs and MOs

5.1 Representing Krn Arrows as AMKs

More precisely, kernels are associated to *abstract Markov kernels* (AMKs for short), which are a generalisation of stochastic matrices. Below, we denote by $\mathbb{1}_X$ the constant function equal to 1 on the space X. Since all measures we consider are finite, $\mathbb{1}_X \in L_p^+(X,\mu)$ for all $p \in [1,\infty]$.

Definition 7 (Abstract Markov kernels). *An* **Lp** *morphism* $A : L_p^+(Y,\nu) \to L_p^+(X,\mu)$ *is an* AMK *if* $A(\mathbb{1}_Y) = \mathbb{1}_X$ *and* $\|A\| = 1$. *Clearly, AMKs are closed under composition and the identity operator is trivially an AMK.* $\mathbf{AMK_p}$ *is the subcategory of* **Lp** *having the same objects and where morphisms are restricted to AMKs.*

Example 4. Let us look at the particular case where X and Y are finite discrete spaces and μ, ν finite measures with full support. Then $L_p^+(X,\mu) \cong \mathbb{R}^X$ and similarly for $L_p^+(Y,\nu)$. Therefore, A corresponds to an $Y \times X$ matrix. The constraint that $A(\mathbb{1}_Y) = \mathbb{1}_X$ amounts to asking that the rows of A sum to 1, i.e. that A is a stochastic matrix.

The adjoint of an AMK is in general *not* an AMK. In the finite case, this reflects the fact that the transpose of a stochastic matrix is not necessarily stochastic. Adjoints of AMKs are called *Markov operators* (MOs for short). Whereas AMKs pull back observables, an MO pushes densities forward. In the following, we make use of the fact that for $p \leq q$, any $u \in L_q^+(X,\mu)$ belongs to $L_p^+(X,\mu)$.

Definition 8 (Markov operators). *An arrow* $A : L_p^+(X,\mu) \to L_p^+(Y,\nu)$ *is an* MO *if for all* $u \in L_p^+(X,\mu)$, $\|A(u)\|_1 = \|u\|_1$ *and* $\|A\| = 1$. $\mathbf{MO_p}$ *is the subcategory of* **Lp** *having the same objects and where morphisms are restricted to MOs.*

Notice that we require an MO to be norm preserving for the L_1^+ norm. This is a mass preservation constraint in disguise. Adjunction maps AMKs to MOs and conversely.

Proposition 3. *The equivalence of categories* $-^* : \mathbf{Lp} \to \mathbf{Lq}^{op}$ *restricts to an equivalence of categories* $-^* : \mathbf{AMK_p} \to \mathbf{MO_q}^{op}$.

We now introduce a family of contravariant functors $\mathsf{T}_p : \mathbf{Krn}^{op} \to \mathbf{AMK_p}$. On objects, we set $\mathsf{T}_p(X,\mu) = L_p^+(X,\mu)$. For $f : (X,\mu) \to (Y,\nu)$ a **Krn** arrow, and for $v \in \mathsf{T}_p(Y,\nu) = L_p^+(Y,\nu)$, we define $\mathsf{T}_p(f)(v)(x) = \int_Y v\, df(x)$. The following theorem generalises the interpretation of kernels as stochastic matrices given in Sect. 2.

Theorem 5. *T_p is a functor from $\mathbf{Krn}^{op}$ to $\mathbf{AMK_p}$.*

The relationship between AMKs and MOs is summed up in Fig. 2. Notice that $\mathbf{AMK_p}$ and $\mathbf{MO_p}$ are subcategories of **Lp** which are not full.

5.2 Bayesian Inversion in Krn

Recall that Theorem 2 gives Bayesian inversion as a bijection

$$-^\dagger : \mathbf{Krn}(X,\mu;Y,\nu)/\mu \cong \mathbf{Krn}(Y,\nu;X,\mu)/\nu.$$

T_p is $\sim$-invariant, which allows us to apply it to $\sim$-equivalence classes of arrows.

Lemma 5. *Let $f, f' : (X,\mu) \to (Y,\nu)$ be such that $f \sim f'$. Then for all $p \in [1,\infty]$, $\mathsf{T}_p(f) = \mathsf{T}_p(f')$.*

Proof. Since $\mu(\{x \mid f(x) \neq f'(x)\}) = 0$, we have for all function $g : \mathsf{G}(Y) \to [0,\infty]$ that $\mu(\{x \mid g \circ f(x) \neq g \circ f'(x)\}) = 0$. Taking $g = ev_v(\lambda) = \int_Y v\, d\lambda$, the sought property follows. □

The following theorem states that pointful Bayesian inversion implements adjunction.

Theorem 6. *For all* **Krn** *arrow $f : (X,\mu) \to (Y,\nu)$ and all Hölder conjugate (p,q), $\mathsf{T}_p(f^\dagger) = \mathsf{T}_q(f)^*$.*

Proof. It is enough to prove that for all $u \in L_p^+(X,\mu), v \in L_q^+(Y,\nu)$, we have $\langle \mathsf{T}_p(f^\dagger)(u), v\rangle_Y = \langle u, \mathsf{T}_q(f)(v)\rangle_X$. We compute:

$$\begin{aligned}
\langle \mathsf{T}_p(f^\dagger)(u), v\rangle_Y &= \int_{y\in Y} v(y) \int_{x\in X} u(x)\, df^\dagger(y)\, d\nu \\
&= \int_{y\in Y} \int_{(x,-)\in X\times Y} u(x)v(y)\, d\pi_Y^\dagger(y)\, d\nu \quad (*) \\
&= \int_{(x,y)\in X\times Y} u(x)v(y)\, dI_{X,\mu}^{Y,\nu}(f) \\
&= \int_{x\in X} \int_{(-,y)\in X\times Y} u(x)v(y)\, d\pi_X^\dagger(x)\, d\mu \\
&= \int_{x\in X} u(x) \int_{y\in Y} v(y)\, df(x)\, d\mu \qquad (*) \\
&= \langle u, \mathsf{T}_q(f)(v)\rangle_X
\end{aligned}$$

This string of equations follows from the definition of $-^\dagger$ (Theorem 2). At the equations marked $(*)$ we used the characteristic property of disintegrations to move u (resp. v) in (resp. out of) the integral (see Theorem 1). □

This proves that Bayesian inversion is really just adjunction. However, performing Bayesian inversion in **Krn** relies on standard Borel assumptions, while adjunction does not! Most importantly, Bayesian inversion in $\omega\mathbf{CC}$ is better structured, as it corresponds to a categorical duality.

6 Pointless Bayesian Inversion Through Couplings

Under standard Borel assumptions, Bayesian models can be given equivalently either in terms of **Krn** arrows or more classically in terms of *joint probabilities* (i.e. couplings). The latter appears crucially in the definition of the pointful inverse, as demonstrated in Theorem 2. However, pointless Bayesian inversion seems *prima facie* to do away with these objects entirely. We conclude this work by shedding some light on the status of couplings w.r.t. pointless inversion: we show that the bijection $I^{Y,\nu}_{X,\mu} : \mathbf{Krn}(X,\mu;Y,\nu)/\mu \to \mathbf{\Gamma}(X,\mu;Y,\nu)$ defined in Sect. 3.3 generalises, for X, Y arbitrary measurable spaces, to a bijection from the Homset $\mathbf{MO}_\infty(L^+_\infty(X,\mu);L^+_\infty(Y,\nu))$ to the set of couplings $\mathbf{\Gamma}(X,\mu;Y,\nu)$. In this setting, we prove that Bayesian inversion amounts to permuting the coordinates of the coupling. Our first ingredient is a map from couplings to ω-continuous linear operators. The key observation is the following.

Proposition 4. *For all $p \in [1,\infty]$, any coupling $\gamma \in \mathbf{\Gamma}(X,\mu;Y,\nu)$ induces an ω-continuous linear operator $\mathsf{K}_p(\gamma) \in \mathbf{AMK_p}(X,\mu;Y,\nu)$ defined for $u \in L^+_p(X,\mu)$ and $v \in L^+_q(Y,\nu)$ (using $\varepsilon_p : L^+_p(Y,\nu) \cong L^{+,*}_q(Y,\nu)$ for (p,q) Hölder conjugate) as $\mathsf{K}_p(\gamma)(u)(v) = \int_{(x,y)\in X\times Y} u(x)v(y)\, d\gamma$.*

Dually to Proposition 4, any MO gives rise to a probability measure (but not necessarily a coupling!). For $A : L^+_p(X,\mu) \to L^+_p(Y,\nu)$ and $B_X \times B_Y$ a basic measurable rectangle in $X \times Y$, we define:

$$\mathsf{C}_p(A)(B_X \times B_Y) = \int_Y \mathbb{1}_{B_Y} A(\mathbb{1}_{B_X})\, d\nu. \tag{5}$$

Lemma 6. *For all MO $A : L^+_p(X,\mu) \to L^+_p(Y,\nu)$, $\mathsf{C}_p(A) \in \mathsf{G}(X \times Y)$.*

It is not obvious what a necessary and sufficient condition should be for $\mathsf{C}_p(A)$ to give rise to a coupling. However, we have the following reasonable sufficient condition.

Proposition 5. *For all MO $A : L^+_\infty(X,\mu) \to L^+_\infty(Y,\nu)$, $\mathsf{C}_\infty(A) \in \mathbf{\Gamma}(X,\mu;Y,\nu)$.*

C and K are the counterparts of respectively I and D in Sect. 3.3, with kernels replaced by respectively MOs and AMKs. However, no quotient is needed to obtain the following result, which states that pointless Bayesian inversion (i.e. adjunction) coincides in the world of couplings to the operation which permutes the coordinates (namely the isomorphism $\mathsf{G}(\sigma) : \mathsf{G}(X \times Y) \to \mathsf{G}(Y \times X)$).

Proposition 6. *For all MO $A : L^+_\infty(X,\mu) \to L^+_\infty(Y,\nu)$, $A^* = \mathsf{K}_1 \circ \mathsf{G}(\sigma) \circ \mathsf{C}_\infty(A)$.*

In order to close the circle, we prove that couplings are indeed in bijections with $\mathbf{MO}_\infty$ arrows (and hence, by duality, $\mathbf{AMK}_1$ arrows).

Theorem 7. *For all X, Y measurable and $\mu \in \mathsf{G}(X), \nu \in \mathsf{G}(Y)$, C_∞ defines a bijection $\mathbf{MO}_\infty(L^+_\infty(X,\mu); L^+_\infty(Y,\nu)) \cong \mathbf{\Gamma}(X,\mu;Y,\nu)$.*

The correspondence between adjunction and the permutation of coupling coordinates together with this last result show that couplings are really at the heart of the semantics of Bayesian inversion.

7 Conclusion

Pointless Bayesian inversion has several qualities which its pointful counterpart lacks: it does not rely on Polish assumptions on the underlying space, it is better typed (as it boils down to an equivalence of categories between abstract Markov kernels and Markov operators) and it admits a trivial and elegant computational interpretation in terms of couplings (as well as the structure of a self-duality on the category of couplings sketched above).

This pointless categorical approach to Bayesian inversion opens the way for exciting new research. First, one yearns to reinterpret previous constructions performed in a kernel-centric way in this new light, such as [12]. Also, the connection between our categories of operators and couplings hints at connections with the Kantorovich distance [13]. For instance, one could study issues of convergence of learning using the weak topology on the space of couplings, which suggests possibly fruitful connections with information geometry.

But chiefly, our more structured framework allows one to reason on the interactions between the approximation of Markov processes by averaging [3] and Bayesian inversion. For instance, we can now ask whether some properties of the Bayesian learning procedure are *profinite*, i.e. entirely characterised by considering the finite approximants (one thinks of issues of convergence of learning, for instance). More generally, we posit that pointless inversion is the right tool to investigate *approximate learning*.

References

1. Ackerman, N.L., Freer, C.E., Roy, D.M.: Noncomputable conditional distributions. In: Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS, Toronto, Ontario, Canada, pp. 107–116, 21–24 June 2011
2. Aliprantis, C., Border, K.: Infinite Dimensional Analysis, vol. 32006. Springer, Heidelberg (1999)
3. Chaput, P., Danos, V., Panangaden, P., Plotkin, G.: Approximating markov processes by averaging. J. ACM 61(1), 45 pages (2014)
4. Clerc, F., Dahlqvist, F., Danos, V., Garnier, I.: Pointless learning (long version) (2017)
5. Culbertson, J., Sturtz, K.: A categorical foundation for Bayesian probability. Appl. Categorical Struct. **22**(4), 647–662 (2012)

6. Dahlqvist, F., Danos, V., Garnier, I., Kammar, O.: Bayesian inversion by omega-complete cone duality (invited paper). In: Desharnais, J., Jagadeesan, R. (eds.) 27th International Conference on Concurrency Theory (CONCUR 2016), vol. 59 of Leibniz International Proceedings in Informatics (LIPIcs), pp. 1:1–1:15. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2016)
7. Giry, M.: A categorical approach to probability theory. In: Banaschewski, B. (ed.) Categorical Aspects of Topology and Analysis. LNM, vol. 915, pp. 68–85. Springer, Heidelberg (1982). doi:10.1007/BFb0092872
8. Kallenberg, O.: Foundations of Modern Probability. Springer, New York (1997)
9. Kechris, A.S.: Classical Descriptive Set Theory. Graduate Text in Mathematics, vol. 156. Springer, New York (1995)
10. Kozen, D.: A probabilistic PDL. In: Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing, STOC 1983, pp. 291–297. ACM, New York (1983)
11. Selinger, P.: Towards a semantics for higher-order quantum computation. In: Proceedings of the 2nd International Workshop on Quantum Programming Languages, TUCS General Publication, vol. 33, pp. 127–143 (2004)
12. Staton, S., Yang, H., Heunen, C., Kammar, O., Wood, F.: Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. CoRR, abs/1601.04943 (2016)
13. Villani, C.: Optimal Transport, Old and New. Grundlehren der mathematischen Wissenschaften. Springer, Heidelberg (2006)

On Higher-Order Probabilistic Subrecursion

Flavien Breuvart[1], Ugo Dal Lago[1,2(✉)], and Agathe Herrou[3]

[1] INRIA, Sophia Antipolis, France
flavien.breuvart@inria.fr
[2] University of Bologna, Bologna, Italy
ugo.dallago@unibo.it
[3] ENS de Lyon, Lyon, France
agathe.herrou@ens-lyon.fr

Abstract. We study the expressive power of subrecursive probabilistic higher-order calculi. More specifically, we show that endowing a very expressive deterministic calculus like Gödel's $\mathbb{T}$ with various forms of probabilistic choice operators may result in calculi which are not equivalent as for the class of distributions they give rise to, although they all guarantee *almost-sure* termination. Along the way, we introduce a probabilistic variation of the classic reducibility technique, and we prove that the simplest form of probabilistic choice leaves the expressive power of $\mathbb{T}$ essentially unaltered. The paper ends with some observations about functional expressivity: expectedly, all the considered calculi represent precisely the functions which $\mathbb{T}$ itself represents.

1 Introduction

Probabilistic models are more and more pervasive in computer science and are among the most powerful modeling tools in many areas like computer vision [20], machine learning [19] and natural language processing [17]. Since the early times of computation theory [8], the very concept of an algorithm has been itself generalised from a purely deterministic process to one in which certain elementary computation steps can have a probabilistic outcome. This has further stimulated research in computation and complexity theory [11], but also in programming languages [21].

Endowing programs with probabilistic primitives (e.g. an operator which models sampling from a distribution) poses a challenge to programming language semantics. Already for a minimal, imperative probabilistic programming language, giving a denotational semantics is nontrivial [16]. When languages also have higher-order constructs, everything becomes even harder [14] to the point of disrupting much of the beautiful theory known in the deterministic case [1]. This has stimulated research on denotational semantics of higher-order probabilistic programming languages, with some surprising positive results coming out recently [4,9].

The authors are partially supported by ANR project 14CE250005 ELICA and ANR project 12IS02001 PACE.

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 370–386, 2017.
DOI: 10.1007/978-3-662-54458-7_22

Not much is known about the expressive power of *probabilistic* higher-order calculi, as opposed to the extensive literature on the same subject about *deterministic* calculi (see, e.g. [23,24]). What happens to the class of representable functions if one enriches, say, a deterministic λ-calculus $\mathbb{X}$ with certain probabilistic choice primitives? Are the expressive power or the good properties of $\mathbb{X}$ somehow preserved? These questions have been given answers in the case in which $\mathbb{X}$ is the pure, untyped, λ-calculus [6]: in that case, universality continues to hold, mimicking what happens in Turing machines [22]. But what if $\mathbb{X}$ is one of the many typed λ-calculi ensuring strong normalisation for typed terms [12]?

Let us do a step back, first: when should a higher-order probabilistic program be considered terminating? The question can be given a satisfactory answer being inspired by, e.g., recent works on probabilistic termination in imperative languages and term rewrite systems [2,18]: one could ask the probability of divergence to be 0, i.e., *almost sure termination*, or the stronger *positive almost sure termination*, in which one requires the average number of evaluation steps to be finite. That almost sure termination is a desirable property, even in a probabilistic setting can be seen in the field of languages like Church and Anglican, in which programs are often assumed to be almost surely terminating, e.g. when doing inference by MH algorithms [13].

In this paper, we initiate a study on the expressive power of terminating higher-order calculi, in particular those obtained by endowing Gödel's $\mathbb{T}$ with various forms of probabilistic choice operators. In particular, three operators will be analysed in this paper:

- A binary probabilistic operator $\oplus$ such that for every pair of terms M, N, the term $M \oplus N$ evaluates to either M or N, each with probability $\frac{1}{2}$. This is a rather minimal option which, however, guarantees universality if applied to the untyped λ-calculus [6] (and, more generally, to universal models of computation [22]).
- A combinator $\mathtt{R}$, which evaluates to any natural number $n \geq 0$ with probability $\frac{1}{2^{n+1}}$. This is the natural generalisation of $\oplus$ to sampling from a distribution having *countable* rather than *finite* support. This apparently harmless generalisation (which is absolutely non-problematic in a universal setting) has dramatic consequences in a subrecursive scenario, as we will discover soon.
- A combinator $\mathtt{X}$ such that for every pair of values V, W, the term $\mathtt{X}\langle V, W\rangle$ evaluates to either W or $V(\mathtt{X}\langle V, W\rangle)$, each with probability $\frac{1}{2}$. The operator $\mathtt{X}$ can be seen as a probabilistic variation on $\mathbb{PCF}$'s fixpoint combinator. As such, $\mathtt{X}$ is potentially problematic to termination, giving rise to infinite trees.

This way, various calculi can be obtained, like $\mathbb{T}^{\oplus}$, namely a minimal extension of $\mathbb{T}$, or the full calculus $\mathbb{T}^{\oplus,\mathtt{R},\mathtt{X}}$, in which the three operators are all available. In principle, the only obvious fact about the expressive power of the above mentioned operators is that both $\mathtt{R}$ and $\mathtt{X}$ are at least as expressive as $\oplus$: binary choice can be easily expressed by either $\mathtt{R}$ or $\mathtt{X}$. Less obvious but still easy to prove is the equivalence between $\mathtt{R}$ and $\mathtt{X}$ in presence of a recursive operator (see Sect. 3.3). But how about, say, $\mathbb{T}^{\oplus}$ vs. $\mathbb{T}^{\mathtt{R}}$?

Traditionally, the expressiveness of such languages is evaluated by looking at the set of functions $f : \mathbb{N} \to \mathbb{N}$ defined by typable programs $M : \mathtt{NAT} \to \mathtt{NAT}$. However, in a probabilistic setting, any program $M : \mathtt{NAT} \to \mathtt{NAT}$ computes a function from natural numbers to *distributions* of natural numbers. In order to fit usual criteria, we need to fix a notion of observation of which there are at least two, corresponding to two randomised programming paradigms, namely *Las Vegas* and *Monte Carlo* observations. The main question, then, consists in understanding how the obtained classes relate to each other, and to the class of $\mathbb{T}$-representable functions. Along the way, however, we manage to understand how to capture the expressive power of probabilistic calculi *per se*. This paper's contributions can be summarised as follows:

- We first take a look at the full calculus $\mathbb{T}^{\oplus,\mathtt{R},\mathtt{X}}$, and prove that it enforces almost-sure termination, namely that the probability of termination of any typable term is 1. This is done by appropriately adapting the well-known reducibility technique [12] to a probabilistic operational semantics. We then observe that while $\mathbb{T}^{\oplus,\mathtt{R},\mathtt{X}}$ cannot be *positively* almost surely terminating, $\mathbb{T}^{\oplus}$ indeed is. This already shows that there must be a gap in expressivity. This is done in Sect. 3.
- In Sect. 4, we look more precisely at the expressive power of $\mathbb{T}^{\oplus}$, proving that the mere presence of probabilistic choice does not add much to the expressive power of $\mathbb{T}$: in a sense, probabilistic choice can be "lifted up" to the ambient deterministic calculus.
- We look at other fragments of $\mathbb{T}^{\oplus,\mathtt{R},\mathtt{X}}$ and at their expressivity. More specifically, we prove that (the equiexpressive) $\mathbb{T}^{\mathtt{R}}$ and $\mathbb{T}^{\mathtt{X}}$ represent precisely what $\mathbb{T}^{\oplus}$ can do *at the limit*, in a sense which will be made precise in Sect. 3. This result, which is the most challenging, is given in Sect. 5.
- Section 6 is devoted to proving that both for *Monte Carlo* and for *Las Vegas* observations, the class of representable functions of $\mathbb{T}^{\mathtt{R}}$ coincides with the $\mathbb{T}$-representable ones.

Due to lack of space, most proofs are elided. An extended version of this paper with more details is available [3].

2 Probabilistic Choice Operators, Informally

Any term of Gödel's $\mathbb{T}$ can be seen as a purely deterministic computational object whose dynamics is finitary, due to the well-known strong normalisation theorem (see, e.g., [12]). In particular, the apparent non-determinism due to multiple redex occurrences is completely harmless because of confluence. In this paper, indeed, we even neglect this problem, and work with a reduction strategy, namely weak call-by-value reduction (keeping in mind that all what we will say also holds in call-by-name). Evaluation of a $\mathbb{T}$-term M of type $\mathtt{NAT}$ can be seen as a finite sequence of terms ending in the normal form $\boldsymbol{n}$ of M (see Fig. 1). More generally, the unique normal form of any $\mathbb{T}$ term M will be denoted as $[\![M]\!]$. Noticeably, $\mathbb{T}$ is computationally very powerful. In particular, the $\mathbb{T}$-representable functions

from $\mathbb{N}$ to $\mathbb{N}$ coincide with the functions which are provably total in Peano's arithmetic [12].

As we already mentioned, the most natural way to enrich deterministic calculi and turn them into probabilistic ones consists in endowing their syntax with one or more probabilistic choice operators. Operationally, each of them models the essentially stochastic process of sampling from a distribution and proceeding depending on the outcome. Of course, one has many options here as for *which* of the various operators to grab. The aim of this work is precisely to study to which extent this choice have consequences on the overall expressive power of the underlying calculus.

Suppose, for example, that $\mathbb{T}$ is endowed with the binary probabilistic choice operator $\oplus$ described in the Introduction, whose evaluation corresponds to tossing a fair coin and choosing one of the two arguments accordingly. The presence of $\oplus$ has indeed an impact on the dynamics of the underlying calculus: the evaluation of any term M is not deterministic anymore, but can be modelled as a finitely branching tree (see, e.g., Fig. 3 for such a tree). The fact that all branches of this tree have finite height (and the tree is thus finite) is intuitive, and a proof of it can be given by adapting the well-known reducibility proof of termination for $\mathbb{T}$. In this paper, we in fact prove much more, and establish that $\mathbb{T}^{\oplus}$ can be embedded into $\mathbb{T}$.

If $\oplus$ is replaced by $\mathtt{R}$, the underlying tree is not finitely branching anymore, but, again, there is not (at least apparently) any infinitely long branch, since each of them can somehow be seen as a $\mathbb{T}$ computation (see Fig. 2 for an example). What happens to the expressive power of the obtained calculus? Intuition tells us that the calculus should not be too expressive viz. $\mathbb{T}^{\oplus}$. If $\oplus$ is replaced by $\mathtt{X}$, on the other hand, the underlying tree *is* finitely branching, but its height can be infinite. Actually, $\mathtt{X}$ and $\mathtt{R}$ are easily shown to be equiexpressive in presence of higher-order recursion, as we show in Sect. 3.3. On the other hand, for $\mathtt{R}$ and $\oplus$, no such encoding is available. Nonetheless, $\mathbb{T}^{\mathtt{R}}$ can still be somehow encoded into $\mathbb{T}$ (the embedding being correct only "at the limit") as we will detail in Sect. 5. From this embedding, we can show that applying Monte Carlo or Las Vegas algorithms to $\mathbb{T}^{\oplus,\mathtt{X},\mathtt{R}}$ do not add any expressive power to that $\mathbb{T}$. This is done in Sect. 6.

3 The Full Calculus $\mathbb{T}^{\oplus,\mathrm{R},\mathrm{X}}$

All along this paper, we work with a calculus $\mathbb{T}^{\oplus,\mathtt{R},\mathtt{X}}$ whose *terms* are the ones generated by the following grammar:

$$M, N, L ::= x \mid \lambda x.M \mid M\ N \mid \langle M, N\rangle \mid \pi_1 \mid \pi_2 \mid \mathtt{rec} \mid \mathbf{0} \mid \mathtt{S} \mid M \oplus N \mid \mathtt{R} \mid \mathtt{X}.$$

Please observe the presence of the usual constructs from the untyped λ-calculus, but also of primitive recursion, constants for natural numbers, pairs, and the three choice operators we have described in the previous sections.

As usual, terms are taken modulo α-equivalence. Terms in which no variable occurs free are said *closed*, and are collected in the set $\mathbb{T}_C^{\oplus,\mathtt{R},\mathtt{X}}$. A *value* is simply a closed term from the following grammar:

$$U, V ::= \lambda x.M \;\Big|\; \langle U, V\rangle \;\Big|\; \pi_1 \;\Big|\; \pi_2 \;\Big|\; \mathtt{rec} \;\Big|\; \mathbf{0} \;\Big|\; \mathtt{S} \;\Big|\; \mathtt{S}\ V \;\Big|\; \mathtt{X}.$$

and the set of values is $\mathbb{T}_V^{\oplus,\mathtt{R},\mathtt{X}}$. *Extended values* are (not necessarily closed) terms generated by the same grammar as values with the addition of variables. Closed terms that are not values are called *reducible* and their set is denoted $\mathbb{T}_R^{\oplus,\mathtt{R},\mathtt{X}}$. The expression $\langle M_1, \ldots, M_n\rangle$ stands for $\langle M_1, \langle M_2, \langle \ldots\rangle\rangle\rangle$. A *context* is a term with a unique hole:

$$C := (\!|\cdot|\!) \;\Big|\; \lambda x.C \;\Big|\; C\ M \;\Big|\; M\ C \;\Big|\; \langle C, M\rangle \;\Big|\; \langle M, C\rangle \;\Big|\; C \oplus M \;\Big|\; M \oplus C.$$

We write $\mathbb{T}_{(\!|\cdot|\!)}^{\oplus,\mathtt{R},\mathtt{X}}$ for the set of all such contexts.

Termination of Gödel's $\mathbb{T}$ is guaranteed by the presence of types, which we also need here. *Types* are expressions generated by the following grammar

$$A, B ::= \mathtt{NAT} \;\Big|\; A \to B \;\Big|\; A \times B.$$

Environmental contexts are expressions of the form $\Gamma = x_1 : A_1, \ldots, x_n : A_n$, while *typing judgments* are of the form $\Gamma \vdash M : A$. *Typing rules* are given in Fig. 5. From now on, only typable terms will be considered. We denote by $\mathbb{T}^{\oplus,\mathtt{R},\mathtt{X}}(A)$ the set of terms of type A, and similarly for $\mathbb{T}_C^{\oplus,\mathtt{R},\mathtt{X}}(A)$ and $\mathbb{T}_V^{\oplus,\mathtt{R},\mathtt{X}}(A)$. We use the shortcut $\boldsymbol{n}$ for values of type $\mathtt{NAT}$: $\mathbf{0}$ is already part of the language of terms, while $\boldsymbol{n} + \mathbf{1}$ is simply $\mathtt{S}\ \boldsymbol{n}$.

3.1 Operational Semantics

While evaluating terms in a deterministic calculus ends up in a *value*, the same process leads to a *distribution* of values when performed on terms in a probabilistic calculus. Formalising all this requires some care, but can be done following one of the many definitions from the literature (e.g., [6]).

Given a countable set X, a *distribution* $\mathcal{L}$ *on* X is a probabilistic subdistribution over elements of X:

$$\mathcal{L}, \mathcal{M}, \mathcal{N} \in \mathfrak{D}(X) = \left\{ f : X \to [0,1] \;\middle|\; \sum_{x \in X} f(x) \leq 1 \right\}.$$

We are especially concerned with distributions over terms here. In particular, a *distribution of type* A is simply an element of $\mathfrak{D}(\mathbb{T}^{\oplus,\mathtt{R},\mathtt{X}}(A))$. The set $\mathfrak{D}(\mathbb{T}_V^{\oplus,\mathtt{R},\mathtt{X}})$ is ranged over by metavariables like $\mathcal{U}, \mathcal{V}, \mathcal{W}$. We will use the pointwise order $\leq$ on distributions, which turns them into an $\omega\mathbf{CPO}$. Moreover, we use the following notation for Dirac's distributions over terms: $\{M\} := \left\{ \begin{matrix} M \mapsto 1 \\ N \mapsto 0 \text{ if } M \neq N \end{matrix} \right\}$. The support of a distribution is indicated as $|\mathcal{M}|$; we also define the reducible and

value supports fragments as $|\mathcal{M}|_R := |\mathcal{M}| \cap \mathbb{T}_R^{\oplus,\mathtt{R},\mathtt{X}}$ and $|\mathcal{M}|_V := |\mathcal{M}| \cap \mathbb{T}_V^{\oplus,\mathtt{R},\mathtt{X}}$. Notions like $\mathcal{M}^R$ and $\mathcal{M}^V$ have an obvious and natural meaning: for any $\mathcal{M} \in \mathfrak{D}(X)$ and $Y \subseteq X$, then $\mathcal{M}^Y(x) = \mathcal{M}(x)$ if $x \in \mathbb{T}_Y^{\oplus,\mathtt{R},\mathtt{X}}$ and $\mathcal{M}^Y(x) = 0$ otherwise.

As syntactic sugar, we use integral notations to manipulate distributions, *i.e.*, for any family of distributions $(\mathcal{N}_M)_{M \in \mathbb{T}^{\oplus,\mathtt{R},\mathtt{X}}} : \mathfrak{D}(\mathbb{T}^{\oplus,\mathtt{R},\mathtt{X}})^{\mathbb{T}^{\oplus,\mathtt{R},\mathtt{X}}}$, the expression $\int_{\mathcal{M}} \mathcal{N}_M.dM$ stands for $\sum_{M \in \mathbb{T}^{\oplus,\mathtt{R},\mathtt{X}}} \mathcal{M}(M) \cdot \mathcal{N}_M$ (by abuse of notation, we may define $\mathcal{N}_M$ only for $M \in |\mathcal{M}|$, since the others are not used anyway). The notation can be easily adapted, e.g., to families of real numbers $(p_M)_{M \in \mathbb{T}^{\oplus,\mathtt{R},\mathtt{X}}}$ and to other kinds of distributions. We indicate as $C(\!|\mathcal{M}|\!)$ the push-forward distribution $\int_{\mathcal{M}} \{C(\!|M|\!)\} dM$ induced by a context C, and as $\sum \mathcal{M}$ the norm $\int_{\mathcal{M}} 1 dM$ of $\mathcal{M}$. Remark, finally, that we have the useful equality $\mathcal{M} = \int_{\mathcal{M}} \{M\} dM$.

Reduction rules of $\mathbb{T}^{\oplus,\mathtt{R},\mathtt{X}}$ are given by Fig. 6. For reasons of simplicity, the relation $\rightarrow$ indicates both a subset of $\mathbb{T}_C^{\oplus,\mathtt{R},\mathtt{X}} \times \mathfrak{D}(\mathbb{T}_C^{\oplus,\mathtt{R},\mathtt{X}})$ *and* a relation on $\mathfrak{D}(\mathbb{T}_C^{\oplus,\mathtt{R},\mathtt{X}}) \times \mathfrak{D}(\mathbb{T}_C^{\oplus,\mathtt{R},\mathtt{X}})$. Notice that the reduction $\rightarrow$ is deterministic. We can easily define $\rightarrow^n$ as the n^{th} exponentiation of $\rightarrow$ and $\rightarrow^*$ as the reflexive and transitive closure of $\rightarrow$ taking the latter as a relation on distributions. In probabilistic systems, we might want to consider infinite reductions such as the ones induced by $\mathtt{X}\langle(\lambda x.x), \mathbf{0}\rangle$, which reduces to $\{\mathbf{0}\}$, but in an infinite number of steps. Remark that for any value V, and whenever $\mathcal{M} \rightarrow \mathcal{N}$, it holds that $\mathcal{M}(V) \leq \mathcal{N}(V)$. As a consequence, we can proceed as follows:

Definition 1. *Let M be a term and let $(\mathcal{M}_n)_{n \in \mathbb{N}}$ be the unique distribution family such that $M \rightarrow^n \mathcal{M}_n$. The* evaluation *of M is the value distribution*

$$[\![M]\!] \;:=\; \{V \mapsto \lim_{n \to \infty} \mathcal{M}_n(V)\} \;\in\; \mathfrak{D}(\mathbb{T}_V^{\oplus,\mathtt{R},\mathtt{X}}).$$

The success *of M is its probability of normalisation, which is formally defined as the norm of its evaluation, i.e., $Succ(M) := \sum [\![M]\!]$. $\mathcal{M}_n^{\Delta V}$ stands for $\{V \mapsto \mathcal{M}_n(V) - \mathcal{M}_{n-1}(V)\}$, the distributions of values reachable in exactly n steps. The* average reduction length *from M is then*

$$[M] \;:=\; \sum_n \left(n \cdot \sum \mathcal{M}_n^{\Delta V}\right) \;\in\; \mathbb{N} \cup \{+\infty\}$$

Notice that, by Rule (r-$\in$), the evaluation is continuous: $[\![\mathcal{M}]\!] = \int_{\mathcal{M}} [\![M]\!] dM$. Any closed term M of type $\mathtt{NAT} \rightarrow \mathtt{NAT}$ *represents* a function $g : \mathbb{N} \rightarrow \mathfrak{D}(\mathbb{N})$ iff for every n, m it holds that $g(n)(m) = [\![M\ \boldsymbol{n}]\!](\boldsymbol{m})$.

3.2 Almost-Sure Termination

We now have all the necessary ingredients to specify a quite powerful notion of probabilistic computation. When, precisely, should such a process be considered *terminating*? Do all probabilistic branches (see Figs. 1, 2, 3 and 4) need to be finite? Or should we stay more liberal? The literature on the subject is pointing to the notion of *almost-sure termination*: a probabilistic computation should be considered terminating if the set of infinite computation branches, although not necessarily empty, has null probability [10,15,18]. This has the following incarnation in our setting:

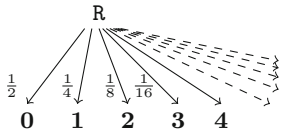


Fig. 1. A reduction in $\mathbb{T}$

Fig. 2. A reduction in $\mathbb{T}^{\mathtt{R}}$

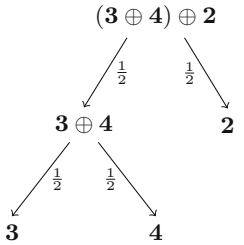


Fig. 3. A reduction in $\mathbb{T}^{\oplus}$

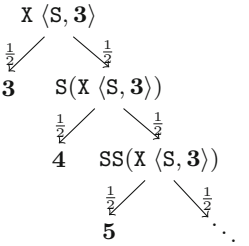


Fig. 4. A reduction in $\mathbb{T}^{\mathtt{X}}$

$$\frac{}{\Gamma, x : A \vdash x : A} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B} \qquad \frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B}$$

$$\frac{}{\Gamma \vdash \mathbf{0} : \mathtt{NAT}} \qquad \frac{}{\Gamma \vdash \mathtt{S} : \mathtt{NAT} \to \mathtt{NAT}} \qquad \frac{}{\Gamma \vdash \mathtt{rec} : A \times (\mathtt{NAT} \to A \to A) \times \mathtt{NAT} \to A}$$

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : B}{\Gamma \vdash \langle M, N \rangle : A \times B} \qquad \frac{}{\Gamma \vdash \pi_1 : (A \times B) \to A} \qquad \frac{}{\Gamma \vdash \pi_2 : (A \times B) \to B}$$

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : A}{\Gamma \vdash M \oplus N : A} \qquad \frac{}{\Gamma \vdash \mathtt{R} : \mathtt{NAT}} \qquad \frac{}{\Gamma \vdash \mathtt{X} : (A \to A) \times A \to A}$$

Fig. 5. Typing rules.

$$\frac{}{(\lambda x.M)\ V \to \left\{ M[V/x] \right\}}\ (r\text{-}\beta) \qquad \frac{M \to \mathcal{M}}{M\ V \to \mathcal{M}\ V}\ (r\text{-}@L)$$

$$\frac{N \to \mathcal{N}}{M\ N \to M\ \mathcal{N}}\ (r\text{-}@R) \qquad \frac{M \to \mathcal{M}}{\langle M, N \rangle \to \langle \mathcal{M}, N \rangle}\ (r\text{-}\langle\cdot\rangle L) \qquad \frac{M \to \mathcal{M}}{\langle V, M \rangle \to \langle V, \mathcal{M} \rangle}\ (r\text{-}\langle\cdot\rangle R)$$

$$\frac{}{\mathtt{rec}\langle U, V, \mathbf{0} \rangle \to \left\{ U \right\}}\ (r\text{-}\mathtt{rec}\mathbf{0}) \qquad \frac{}{\mathtt{rec}\langle U, V, \mathtt{S}\ \boldsymbol{n} \rangle \to \left\{ V\ \boldsymbol{n}\ (\mathtt{rec}\langle U, V, \boldsymbol{n} \rangle) \right\}}\ (r\text{-}\mathtt{rec}\mathtt{S})$$

$$\frac{}{\pi_1\ \langle V, U \rangle \to \left\{ V \right\}}\ (r\text{-}\pi_1) \qquad \frac{}{\pi_2\ \langle V, U \rangle \to \left\{ U \right\}}\ (r\text{-}\pi_2) \qquad \frac{}{\mathtt{R} \to \left\{ \boldsymbol{n} \mapsto \frac{1}{2^{n+1}} \right\}_{n \in \mathbb{N}}}\ (r\text{-}\mathtt{R})$$

$$\frac{}{M \oplus N \to \left\{ \begin{matrix} M \mapsto \frac{1}{2} \\ N \mapsto \frac{1}{2} \end{matrix} \right\}}\ (r\text{-}\oplus) \qquad \frac{}{\mathtt{X}\langle V, W \rangle \to \left\{ \begin{matrix} V\ (\mathtt{X}\ \langle V, W \rangle) \mapsto \frac{1}{2} \\ W \qquad\qquad\quad \mapsto \frac{1}{2} \end{matrix} \right\}}\ (r\text{-}\mathtt{X})$$

$$\frac{\forall M \in |\mathcal{M}|_R,\ M \to \mathcal{N}_M \qquad \forall V \in |\mathcal{M}|_V,\ \mathcal{N}_V = \{V\}}{\mathcal{M} \to \int_{\mathcal{M}} \mathcal{N}_M.d\mathcal{M}}\ (r\text{-}\in)$$

Fig. 6. Operational semantics.

Definition 2. *A term M is said to be* almost-surely terminating *(AST) iff $Succ(M) = 1$.*

This section is concerned with proving that $\mathbb{T}^{\oplus,\mathtt{R},\mathtt{X}}$ indeed guarantees almost-sure termination. This will be done by adapting Girard-Tait's reducibility technique.

The following is a crucial intermediate step towards Theorem 1, the main result of this section.

Lemma 1. *For any M, N, it holds that $[\![M\ N]\!] = [\![[\![M]\!]\ [\![N]\!]]\!]$. In particular, if the application $M\ N$ is almost-surely terminating, so are M and N.*

Theorem 1. *The full system $\mathbb{T}^{\oplus,\mathtt{R},\mathtt{X}}$ is almost-surely terminating (AST),* i.e.*,*

$$\forall M \in \mathbb{T}^{\oplus,\mathtt{R},\mathtt{X}}, \quad Succ(M) = 1.$$

Proof. The proof[1] is based on the notion of a *reducible* term which is given as follows by induction on the structure of types:

$$\begin{aligned} Red_{\mathtt{NAT}} &:= \left\{M \in \mathbb{T}^{\oplus,\mathtt{R},\mathtt{X}}(\mathtt{NAT}) \mid M \text{ is AST}\right\}; \\ Red_{A\to B} &:= \{M \mid \forall V \in Red_A \cap \mathbb{T}_V^{\oplus,\mathtt{R},\mathtt{X}},\ (M\ V) \in Red_B\}; \\ Red_{A\times B} &:= \{M \mid (\pi_1\ M) \in Red_A,\ (\pi_2\ M) \in Red_B\}. \end{aligned}$$

Then we can observe that:

- *The reducibility candidates over Red_A are $\to$-saturated:* by induction on A we can indeed show that if $M \to \mathcal{M}$ then $|\mathcal{M}| \subseteq Red_A$ iff $M \in Red_A$.
- *The reducibility candidates over Red_A are precisely the AST terms M such that $[\![M]\!] \subseteq Red_A$:* this goes by induction on A. Trivial for $A = \mathtt{NAT}$. Let $M \in Red_{B\to C}$: remark that there is a value $V \in Red_B$, thus $(M\ V) \in Red_C$ and $(M\ V)$ is AST by IH; using Lemma 1 we get M AST and it is easy to see that if $U \in |[\![M]\!]|$ then $U \in |\mathcal{M}|$ for some $M \to^* \mathcal{M}$ so that $U \in Red_{B\to C}$ by saturation. Conversely, let M be AST with $|[\![M]\!]| \subseteq Red_{B\to C}$ and let $V \in Red_B$ be a value: by IH, for any $U \in |[\![M]\!]| \subseteq Red_{B\to C}$ we have $(U\ V)$ AST with an evaluation supported by elements of Red_C; by Lemma 1 $[\![M\ V]\!] = [\![[\![M]\!]\ V]\!]$ meaning that $(M\ V)$ is AST and has an evaluation supported by elements of Red_C, so that we can conclude by IH. Similar for products.
- *Every term M such that $x_1 : A_1, \ldots, x_n : A_n \vdash M : B$ is a candidate in the sense that if $V_i \in Red_{A_i}$ for every $1 \le i \le n$, then $M[V_1/x_1, \ldots, V_n/x_n] \in Red_B$:* by induction on the type derivation. The only difficult cases are those for the application and for $\mathtt{X}$ (the one for $\mathtt{rec}$ is just an induction on its third argument).
  - We need to show that if $M \in Red_{A\to B}$ and $N \in Red_A$ then $(M\ N) \in Red_B$. But since $N \in Red_A$, this means that it is AST and for every $V \in |[\![N]\!]|$, $(M\ V) \in Red_B$. In particular, by Lemma 1, we have $[\![M\ N]\!] = [\![M\ [\![N]\!]]\!]$ so that $(M\ N)$ is AST and $|[\![M\ N]\!]| \subseteq \bigcup_{V\in|[\![N]\!]|} |[\![M\ V]\!]| \subseteq Red_B$.

[1] Another proof of almost sure termination using reducibility candidate can be found in [25].

- We need to show that for any value $U \in Red_{A\to A}$ and $V \in Red\,A$ if holds that $(\mathtt{X}\ \langle U, V\rangle) \in Red_A$. By an easy induction on n, $(U^n\ V) \in Red_A$. Moreover, by an easy induction on n we have $[\![\mathtt{X}\,\langle U, V\rangle]\!] = \frac{1}{2^{n+1}}[\![U^n\ (\mathtt{X}\ \langle U, V\rangle)]\!] + \sum_{i\leq n} \frac{1}{2^{i+1}}[\![U^i\ V]\!]$ so that at the limit $[\![\mathtt{X}\,\langle U, V\rangle]\!] = \sum_{i\in\mathbb{N}} \frac{1}{2^{i+1}}[\![U^i\ V]\!]$. We can then conclude that $(\mathtt{X}\ \langle U, V\rangle)$ is AST (since each of the $(U^i\ V) \in Red_B$ are AST and $\sum_i \frac{1}{2^{i+1}} = 1$) and that $|[\![M\ N]\!]| = \bigcup_i |[\![U^i\ V]\!]| \subseteq Red_A$.

This concludes the proof. □

Almost-sure termination could however be seen as too weak a property: there is no guarantee about the average computation length. For this reason, another stronger notion is often considered, namely *positive* almost-sure termination:

Definition 3. *A term M is said to be* positively almost-surely terminating *(or* PAST*) iff the average reduction length $[M]$ is finite.*

Gödel's $\mathbb{T}$, when paired with $\mathtt{R}$, is combinatorially too powerful to guarantee positive almost sure termination:

Theorem 2. *$\mathbb{T}^{\oplus,\mathtt{R},\mathtt{X}}$ is not positively almost-surely terminating.*

Proof. The naive exponential function applied to $\mathtt{R}$ is computing, with probability $\frac{1}{2^{n+1}}$ the number 2^{n+1} in time 2^{n+1}. This is already a counterexample, because it clearly has infinite average termination time. □

3.3 On Fragments of $\mathbb{T}^{\oplus,\mathrm{R},\mathrm{X}}$: A Roadmap

The calculus $\mathbb{T}^{\oplus,\mathtt{R},\mathtt{X}}$ contains at least four fragments, namely Gödel's $\mathbb{T}$ and the three fragments $\mathbb{T}^{\oplus}$, $\mathbb{T}^{\mathtt{R}}$ and $\mathbb{T}^{\mathtt{X}}$ corresponding to the three probabilistic choice operators we consider. It is then natural to ask how these fragments relate to each other as for their respective expressive power. At the end of this paper, we will have a very clear picture in front of us.

The first result we can give is the equivalence between the apparently dual fragments $\mathbb{T}^{\mathtt{R}}$ and $\mathbb{T}^{\mathtt{X}}$. The embeddings are in fact quite simple:

Proposition 1. *$\mathbb{T}^{\mathtt{R}}$ and $\mathbb{T}^{\mathtt{X}}$ are both equiexpressive with $\mathbb{T}^{\oplus,\mathtt{R},\mathtt{X}}$.*

Proof. The calculus $\mathbb{T}^{\mathtt{R}}$ embeds the full system $\mathbb{T}^{\oplus,\mathtt{R},\mathtt{X}}$ via the encoding:[2]

$$M \oplus N := \mathtt{rec}\langle \lambda z.N, \lambda xyz.M, \mathtt{R}\rangle \mathbf{0}; \qquad \mathtt{X} := \lambda xy.\mathtt{rec}\langle y, \lambda z.x, \mathtt{R}\rangle.$$

The fragment $\mathbb{T}^{\mathtt{X}}$ embeds the full system $\mathbb{T}^{\oplus,\mathtt{R},\mathtt{X}}$ via the encoding:

$$M \oplus N := \mathtt{X}\langle \lambda xy.M, \lambda y.N\rangle\ \mathbf{0}; \qquad \mathtt{R} := \mathtt{X}\langle \mathtt{S}, \mathbf{0}\rangle.$$

In both cases, the embedding is compositional and preserves types. That the two embeddings are correct can be proved easily, see [3]. □

[2] Notice that the dummy abstractions on z and the $\mathbf{0}$ at the end ensure the correct reduction order by making $\lambda z.N$ a value.

Notice how simulating $\mathtt{X}$ by $\mathtt{R}$ requires the presence of recursion, while the converse is not true. The implications of this fact are intriguing, but lie outside the scope of this work.

In the following, we will no longer consider $\mathbb{T}^{\mathtt{X}}$ nor $\mathbb{T}^{\oplus,\mathtt{R},\mathtt{X}}$ but only $\mathbb{T}^{\mathtt{R}}$, keeping in mind that all these are equiexpressive due to Proposition 1. The rest of this paper, thus, will be concerned with understanding the relative expressive power of the three fragments $\mathbb{T}$, $\mathbb{T}^{\oplus}$, and $\mathbb{T}^{\mathtt{R}}$. Can any of the (obvious) strict *syntactical* inclusions between them be turned into a strict *semantic* inclusion? Are the three systems equiexpressive?

In order to compare probabilistic calculi to deterministic ones, several options are available. The most common one is to consider notions of observations over the probabilistic outputs; this will be the purpose of Sect. 6. In this section, we will look at whether it is possible to deterministically represent the distributions computed by the probabilistic calculus at hand. We say that the distribution $\mathcal{M} \in \mathfrak{D}(\mathbb{N})$ is *finitely represented* by[3] $f : \mathbb{N} \to \mathbb{B}$, if there exists a q such that for every $k \geq q$ it holds that $f(k) = 0$ and

$$\mathcal{M} = \{\boldsymbol{k} \mapsto f(k)\}.$$

Moreover, the definition can be extended to families of distributions $(\mathcal{M}_n)_n$ by requiring the existence of $f : \mathbb{N} \times \mathbb{N} \to \mathbb{B}$, $q : \mathbb{N} \to \mathbb{N}$ such that for all $k \geq q(n)$, $f(n,k) = 0$ and

$$\forall n, \quad \mathcal{M}_n = \{\boldsymbol{k} \mapsto f(n,k)\}.$$

In this case, we say that the representation is *parameterized*.

We will see in Sect. 4 that the distributions computed by $\mathbb{T}^{\oplus}$ are exactly the (parametrically) finitely representable by $\mathbb{T}$ terms. In $\mathbb{T}^{\mathtt{R}}$, however, distributions are more complex (infinite, non-rational). That is why only a characterisation in terms of approximations is possible. More specifically, a distribution $\mathcal{M} \in \mathfrak{D}(\mathbb{N})$ is said to be *functionally represented* by two functions $f : \mathbb{N} \times \mathbb{N} \to \mathbb{B}$ and $g : \mathbb{N} \to \mathbb{N}$ iff for every $n \in \mathbb{N}$ and for every $k \geq g(n)$ it holds that $f(n,k) = 0$ and

$$\sum_{k \in \mathbb{N}} \left| \mathcal{M}(k) - f(n,k) \right| \leq \frac{1}{n}.$$

In other words, the distribution $\mathcal{M}$ can be approximated arbitrarily well, and uniformly, by finitely representable ones. Similarly, we can define a parameterised version of this definition at first order.

In Sect. 5, we show that distributions generated by $\mathbb{T}^{\mathtt{R}}$ terms are indeed uniform limits over those of $\mathbb{T}^{\oplus}$; using our result on $\mathbb{T}^{\oplus}$ this give their (parametric) functional representability in the deterministic $\mathbb{T}$.

[3] Here $\mathbb{B}$ stands for the set of dyadic numbers, i.e. rationals in the form $\frac{n}{2^m}$ (where $m, n \in \mathbb{N}$) and $\mathtt{BIN}$ for their representation in system $\mathbb{T}$, encoded as pairs of natural numbers.

4 Binary Probabilistic Choice

This section is concerned with two theoretical results on the expressive power of $\mathbb{T}^{\oplus}$. The main feature of $\mathbb{T}^{\oplus}$ is that its terms are *positively* almost surely terminating. This is a corollary of the following theorem (whose proof [3] proceeds again by reducibility).

Theorem 3. *For any term $M \in \mathbb{T}^{\oplus}$, $M \to^* [\![M]\!]$.*

Now, if $M \to^n [\![M]\!]$, then $[M]$ can be at most n since the distribution $\mathcal{M}_m^{\Delta V}$ of values reachable in exactly m steps (see Definition 1) will be 0 for every $m > n$. But this means that typable terms normalise in finite time:

Corollary 1. *Any term $M \in \mathbb{T}^{\oplus}$ is positively almost-surely terminating.*

But this is not the only consequence. In fact, the finiteness of $[\![M]\!]$ and the fact that $\mathbb{T}^{\oplus}$ is sufficiently expressive allow for a finite representation of $\mathbb{T}^{\oplus}$-distributions by $\mathbb{T}$-definable functions. To prove it, let us consider an extension of $\mathbb{T}$ with a single memory-cell c of type $\mathtt{NAT}$. This memory-cell is used to store some "random coins" simulating probabilistic choices. The operator $\oplus$ can be encoded as follows:

$$(M \oplus N)^* \quad := \quad \mathtt{if}\ (\mathtt{mod}_2\ c)\ \mathtt{then}\ (c{:=}\mathtt{div}_2\ c\,;M^*)\ \mathtt{else}\ (c{:=}\mathtt{div}_2\ c\,;N^*)$$

Notice that conditionals and modulo arithmetic are easily implementable in $\mathbb{T}$. From Theorem 3, we know that for any $M \in \mathbb{T}^{\oplus}(\mathtt{NAT})$, there is $n \in \mathbb{N}$ such that $M \to^n [\![M]\!]$, and since the evaluation of M can thus involve at most n successive probabilistic choices, we have that

$$[\![M]\!](\boldsymbol{k}) = \frac{\#\{m < 2^n \mid k = [\![c{:=}\boldsymbol{m}\,;M^*]\!]\}}{2^n}.$$

By way of a state-passing transformation, we can enforce $(c{:=}\boldsymbol{m}\,;M^*)$ into a term of $\mathbb{T}$. But then, the whole $\#\{m{<}2^n \mid k{=}[\![c{:=}\boldsymbol{m}\,;M^*]\!]\}$ can be represented as a $\mathbb{T}$-term $k : \mathbb{N} \vdash N : \mathbb{N}$ which finitely represents the distribution $[\![M]\!]$.

In the long version of this paper [3], a stronger result is proved, namely that for any functional $M \in \mathbb{T}^{\oplus}(\mathtt{NAT}{\to}\mathtt{NAT})$, there are terms $M_{\downarrow} \in \mathbb{T}(\mathtt{NAT}{\to}\mathtt{NAT}{\to}\mathtt{NAT})$ and $M_{\#} \in \mathbb{T}(\mathtt{NAT}{\to}\mathtt{NAT})$ such that for all $n \in \mathbb{N}$:

$$[\![M\ \boldsymbol{n}]\!](\boldsymbol{k}) \quad = \quad \frac{\#\{m < 2^{[\![M_{\#}\ \boldsymbol{n}]\!]} \mid k = [\![M_{\downarrow}\ \boldsymbol{n}\ \boldsymbol{m}]\!]\}}{2^{[\![M_{\#}\ \boldsymbol{n}]\!]}}.$$

The supplementary difficulty, here, comes from the bound $M_{\#}$ that have to be computed dynamically as it depends on its argument $\boldsymbol{n}$.

As a consequence:

Theorem 4. *Distributions generated by $\mathbb{T}^{\oplus}$-terms are precisely those which can be finitely generated by parameterized $\mathbb{T}$-functionals;* i.e., *for any term $M : \mathtt{NAT} \to \mathtt{NAT}$, there are two $\mathbb{T}$-functionals $f : (\mathbb{N} \times \mathbb{N}) \to \mathbb{B}$ and $q : \mathbb{N} \to \mathbb{N}$ such that for all n:*

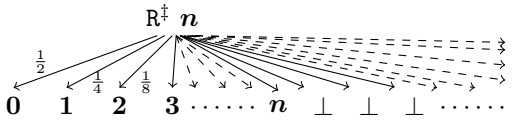
$$[\![M\boldsymbol{n}]\!] = \{\boldsymbol{k} \mapsto f(n,k) \mid k \leq q(n)\}.$$

5 Countable Probabilistic Choice

In this section, we show that $\mathbb{T}^{\oplus}$ approximates $\mathbb{T}^{\mathtt{R}}$: for any term $M \in \mathbb{T}^{\mathtt{R}}(\mathtt{NAT})$, there is a term $N \in \mathbb{T}^{\oplus}(\mathtt{NAT} \to \mathtt{NAT})$ that represents a sequence approximating M uniformly. We will here make strong use of the fact that M has type $\mathtt{NAT}$. This is a natural drawback when we understand that the encoding $(\cdot)^{\dagger}$ on which the result above is based is not direct, but goes through yet another state passing style transformation. Nonetheless, everything can be lifted easily to the first order, achieving the parameterisation of our theorem.

The basic idea behind the embedding $(\cdot)^{\dagger}$ is to mimic any instance of the operator $\mathtt{R}$ in the source term by some term $\mathbf{0} \oplus (1 \oplus (\cdots (\boldsymbol{n} \oplus \bot) \cdots))$, where n is *sufficiently large*, and $\bot$ is an arbitrary value of type $\mathtt{NAT}$. Of course, the semantics of this term is *not* the same as that of $\mathtt{R}$, due to the presence of $\bot$; however, n will be chosen sufficiently large for the difference to be negligible. Notice, moreover, that this term can be generalized into the following parametric form $\mathtt{R}^{\ddagger} := \lambda x.\mathtt{rec}\ \langle \bot, (\lambda x.\mathtt{S} \oplus (\lambda y.\mathbf{0})), x \rangle$.

Once $\mathtt{R}^{\ddagger}$ is available, a natural candidate for the encoding $(\cdot)^{\dagger}$ would be to consider something like $M^{\ddagger} := \lambda z.M[(\mathtt{R}^{\ddagger}\ z)/\mathtt{R}]$. In the underlying execution tree, $(M^{\ddagger}\ \boldsymbol{n})$ correctly simulates the first n branches of $\mathtt{R}$ (which has infinite arity), but truncates the rest with garbage terms $\bot$:



The question is whether the remaining untruncated tree has a "sufficient weight", i.e., whether there is a minimal bound to the probability to stay in this untruncated tree. However, in general $(\cdot)^{\ddagger}$ fails on this point, not achieving to approximate M uniformly. In fact, this probability is basically $(1 - \frac{1}{2^n})^d$ where d is its depth. Since in general the depth of the untruncated tree can grow very rapidly on n in a powerful system like $\mathbb{T}$, there is no hope for this transformation to perform a uniform approximation.

The solution we are using is to have the precision m of $\mathbf{0} \oplus (\mathbf{1} \oplus (\cdots (\boldsymbol{m} \oplus \bot) \cdots))$ to *dynamically grow* along the computation. More specifically, in the approximants $M^{\dagger}\ \boldsymbol{n}$, the growing speed of m will increase with n: in the n-th approximant $M^{\dagger}\ \boldsymbol{n}$, the operator $\mathtt{R}$ will be simulated as $\mathbf{0} \oplus (1 \oplus (\cdots (\boldsymbol{m} \oplus \bot) \cdots))$ *and*, somehow, m will be updated to $m + n$. Why does it work? Simply because even for an (hypothetical) infinite and complete execution tree of M, we would stay inside the n^{th} untruncated tree with probability $\prod_{k \geq 0}(1 - \frac{1}{2^{k*n}})$ which is asymptotically above $(1 - \frac{1}{n})$.

Implementing this scheme in $\mathbb{T}^{\oplus}$ requires a feature which is not available (but which can be encoded), namely ground-type references. We then prefer to show that the just described scheme can be realised in an intermediate language called $\mathbb{T}^{\bar{\mathtt{R}}}$, whose operational semantics is formulated not on *terms*, but rather on triples in the form (M, m, n), where M is the term currently being evaluated,

m is the current approximation threshold value, and n is the value of which m is incremented whenever $\mathtt{R}$ is simulated. The operational semantics is standard, except for the following rule:

$$\frac{}{(\bar{\mathtt{R}}, m, n) \to \left\{(k, m{+}n, n) \mapsto \frac{1}{2^{k+1}} \mid k < m\right\}}\ (r\text{-}\bar{\mathtt{R}})$$

Notice how this operator behaves similarly to $\mathtt{R}$ with the exception that it fails when drawing too big of a number (*i.e.*, bigger that the fist state m). Notice that the failure is represented by the fact that the resulting distribution does not necessarily sum to 1. The intermediate language $\mathbb{T}^{\bar{\mathtt{R}}}$ is able to approximate $\mathbb{T}^{\mathtt{R}}$ at every order (Theorem 5 below). Moreover, the two memory cells can be shown to be expressible in $\mathbb{T}^{\oplus}$, again by way of a continuation-passing transformation. Crucially, the initial value of n can be passed as an argument to the encoded term.

For any $M \in \mathbb{T}^{\mathtt{R}}$ we denote $M^* := M[\bar{\mathtt{R}}/\mathtt{R}]$. We say that $(M, m, n) \in \mathbb{T}^{\bar{\mathtt{R}}}$ if $m, n \in \mathbb{N}$ and $M = N^*$ for some $N \in \mathbb{T}^{\mathtt{R}}$. Similarly, $\mathfrak{D}(\mathbb{T}^{\bar{\mathtt{R}}})$ is the set of probabilistic distributions over $\mathbb{T}^{\bar{\mathtt{R}}} \times \mathbb{N}^2$, *i.e.*, over the terms plus states.

For any m and n, the behaviour of M and (M^*, m, n) are similar, except that (M^*, m, n) will "fail" more often. In other words, all $(M^*, m, n)_{m,n\in\mathbb{N}}$ somehow approximate M from below:

Lemma 2. *For any $M \in \mathbb{T}^{\mathtt{R}}$ and any $m, n \in \mathbb{N}$, $[\![M]\!] \succeq [\![M^*, m, n]\!]$, i.e., for every $V \in \mathbb{T}^{\mathtt{R}}_V$, we have*

$$[\![M]\!](V) \geq \sum_{p,q} [\![M^*, m, n]\!](V^*, p, q).$$

Proof. By an easy induction, one can show that for any $\mathcal{M} \in \mathfrak{D}(\mathbb{T}^{\mathtt{R}})$ and $\mathcal{N} \in \mathfrak{D}(\mathbb{T}^{\bar{\mathtt{R}}})$ if $\mathcal{M} \succeq \mathcal{N}$, $\mathcal{M} \to \mathcal{L}$ and $\mathcal{N} \to \mathcal{P}$, then $\mathcal{L} \succeq \mathcal{P}$. This ordering is then preserved at the limit so that we get our result. □

In fact, the probability of "failure" of any $(M, m, n)_{m,n\in\mathbb{N}}$ can be upper-bounded explicitly. More precisely, we can find an infinite product underapproximating the success rate of (M, m, n) by reasoning inductively over the execution $(M, m, n) \to^* [\![(M, m, n)]\!]$, which is possible because of the PAST.

Lemma 3. *For any $M \in \mathbb{T}^{\bar{\mathtt{R}}}$ and any $m, n \geq 1$*

$$Succ(M, m, n) \geq \prod_{k\geq 0} \left(1 - \frac{1}{2^{m+kn}}\right).$$

Proof. We denote $\#(m, n) := \prod_{k\geq 0} \left(1 - \frac{1}{2^{m+kn}}\right)$ and $\#\mathcal{M} := \int_{\mathcal{M}} \#(m, n) dM$ *dmdn*. Remark that for any M and any m, n, if $(M, m, n) \to \mathcal{M}$ then $\mathcal{M}$ is either of the form $\{(N, m, n)\}$ or $\{(N_i, m{+}n, n) \mapsto \frac{1}{2^{i+1}} \mid i < m\}$ for some N of $(N_i)_{i\leq m}$.

Thus we have that if $(M,m,n) \to \mathcal{N}$ then $\#\mathcal{N} = \#(m,n)$ and that if $\mathcal{M} \to \mathcal{N}$ then $\#\mathcal{N} = \#\mathcal{M}$. In particular, since $(M,m,n) \to^* [\![M,m,n]\!]$ we can conclude

$$Succ(M,m,n) \;=\; \int_{[\![M,m,n]\!]} 1\; dMdmdn \;\geq\; \#[\![M]\!] \;=\; \#(m,n) \;=\; \prod_{k\geq 0}\Big(1 - \frac{1}{2^{m+kn}}\Big).$$

□

This gives us an analytic lower bound to the success rate of (M,m,n). However, it is not obvious that this infinite product is an interesting bound, it is not even clear that it can be different from 0. This is why we will further underapproximate this infinite product to get a simpler expression whenever $m = n$:

Lemma 4. *For any $M \in \mathbb{T}^{\bar{\mathtt{R}}}$ and any $n \geq 4$*

$$Succ(M,n,n) \quad \geq \quad 1 - \frac{1}{n}.$$

Proof. By Lemma 3 we have that $Succ(M,n,n) \geq \prod_{k\geq 1}\big(1 - \frac{1}{2^{kn}}\big)$ which is above the product $\prod_{k\geq 1}\big(1 - \frac{1}{n^2k^2}\big)$ whenever $n \geq 4$. This infinite product has been shown by Euler to be equal to $\frac{sin(\frac{\pi}{n})}{\frac{\pi}{n}}$. By an easy numerical analysis we then obtain that $\frac{sin(\frac{\pi}{n})}{\frac{\pi}{n}} \geq 1 - \frac{1}{n}$. □

This lemma can be restated by saying that the probability of "failure" of (M^*,n,n), *i.e.* the difference between $[\![M^*,n,n]\!]$ and $[\![M]\!]$, is bounded by $\frac{1}{n}$. With this we then get our first theorem, which is the uniform approximability of elements of $\mathbb{T}^{\mathtt{R}}$ by those of $\mathbb{T}^{\bar{\mathtt{R}}}$:

Theorem 5. *For any $M \in \mathbb{T}^{\mathtt{R}}$ and any $n \in \mathbb{N}$,*

$$\sum_V \Big|\; [\![M]\!](V) \;-\; \Sigma_{m',n'}[\![M^*,n,n]\!](V^*,m',n') \;\Big| \quad \leq \quad \frac{1}{n}.$$

Proof. By Lemma 2, for each V the difference is positive, thus we can remove the absolute value and distribute the sum. We conclude by using the fact that $Succ(M) = 1$ and $Succ(M^*,n,n) \geq 1 - \frac{1}{n}$. □

The second theorem, *i.e.*, the uniform approximability of ground elements of $\mathbb{T}^{\mathtt{R}}$ by those of $\mathbb{T}^{\oplus}$, follows immediately:

Theorem 6. *Distributions in $\mathbb{T}^{\mathtt{R}}(\mathtt{NAT})$ can be approximated by $\mathbb{T}^{\oplus}$-distributions (which are finitely $\mathbb{T}$-representable), i.e., for any $M \in \mathbb{T}^{\mathtt{R}}(\mathtt{NAT})$, there is $M^\dagger \in \mathbb{T}^{\oplus}(\mathtt{NAT} \to \mathtt{NAT})$ such that:*

$$\forall n, \quad \sum_{\boldsymbol{k}} \Big|\; [\![M]\!](\boldsymbol{k}) \;-\; [\![M^\dagger\; \boldsymbol{n}]\!](\boldsymbol{k}) \;\Big| \quad \leq \quad \frac{1}{n}.$$

Moreover:

- *the encoding is parameterisable, in the sense that for all* $M \in \mathbb{T}^{\mathtt{R}}(\mathtt{NAT} \to \mathtt{NAT})$, *there is* $M^\wr \in \mathbb{T}^{\oplus}(\mathtt{NAT} \to \mathtt{NAT} \to \mathtt{NAT})$ *such that* $[\![(M\ \boldsymbol{m})^\dagger]\!] = [\![M^\wr\ \boldsymbol{m}]\!]$ *for all* $m \in \mathbb{N}$*;*
- *the encoding is such that* $[\![M]\!](\boldsymbol{k}) \leq [\![M^\dagger\ \boldsymbol{n}]\!](\boldsymbol{k})$ *only if* $k = 0$.

Proof. It is clear that in an extension of $\mathbb{T}^{\oplus}$ with two global memory cells m, n and with an exception monad, the $\bar{\mathtt{R}}$ operator can be encoded by the term $\bar{\mathtt{R}} := \mathtt{rec}\ \langle \bot, (\lambda x.\mathtt{S} \oplus (\lambda y.\mathbf{0})), (m\ :=!m{+}!n)\rangle$, where $\bot$ is raising an error/exception and where $m :=!m{+}!n$ is returning the value of m before changing the memory cell to $m + n$. We can conclude by referring to the usual state passing style encoding of exceptions and state-monads into $\mathbb{T}$ (and thus into $\mathbb{T}^{\oplus}$). □

6 Subrecursion

If one wishes to define $\mathbb{T}^{\oplus}$-definable or $\mathbb{T}^{\mathtt{R}}$-definable functions as a set of ordinary set-theoretic functions (say from $\mathbb{N}$ to $\mathbb{N}$), it is necessary to collapse the random output into a deterministic one. As already acknowledged by the complexity community, there are at least two reasonable ways to do so: by using either Monte Carlo or Las Vegas observations.

As the careful reader may have foreseen, the finite (parametric) representation of $\mathbb{T}^{\oplus}$-distributions into $\mathbb{T}$ is collapsing both observations into $\mathbb{T}$-definable functions. One only need to explore the finite representation, the resulting process suffers from an exponential blow-up, which is easily absorbed by $\mathbb{T}$, in which all elementary functions (and much more than that!) can be expressed.

Theorem 7 (Monte Carlo). *Let* $f : \mathbb{N} \to \mathbb{N}$ *and* $M : \mathtt{NAT} \to \mathtt{NAT}$ *a* $\mathbb{T}^{\mathtt{R}}$*-term such that* $(M\ \boldsymbol{m})$ *evaluates into* $f(m)$ *with probability* $p \geq \frac{1}{2} + \frac{1}{g(m)}$ *for a* $\mathbb{T}$*-definable function* g*. Then* f *is* $\mathbb{T}$*-definable.*

Theorem 8 (Las Vegas). *Let* $f : \mathbb{N} \to \mathbb{N}$ *and* $M : \mathtt{NAT} \to \mathtt{NAT}$ *a* $\mathbb{T}^{\mathtt{R}}$*-term such that* $(M\ \boldsymbol{m})$ *evaluate either to* $\mathbf{0}$ *(representing a failure) or to* $\boldsymbol{f(m)+1}$*, the later happening with probability* $p \geq \frac{1}{g(m)}$ *for some* $\mathbb{T}$*-definable function* g*. Then* f *is* $\mathbb{T}$*-definable.*

7 Conclusions

This paper is concerned with the impact of adding various forms of probabilistic choice operators to a higher-order subrecursive calculus in the style of Gödel's $\mathbb{T}$. The presented results help in understanding the relative expressive power of various calculi which can be obtained this way, by showing some separation and equivalence results.

The probabilistic choice operators we consider here are just examples of how one can turn a deterministic calculus like $\mathbb{T}$ into a probabilistic model of computation. The expressiveness of $\mathbb{T}^{\oplus,\mathtt{R},\mathtt{X}}$ is sufficient to encode most reasonable probabilistic operators, but what can we say about their own expressive power? For example, what about a ternary operator in which either of the first two operators is chosen with a probability *which depends* on the value of the third operator? A general theory of probabilistic choice operators and of their expressive power is still lacking.

Another research direction to which this paper hints at consists in studying the logical and proof-theoretical implications of endowing a calculus like $\mathbb{T}$ with probabilistic choice operators. What is even more exciting, however, is the application of the ideas presented here to polynomial time computation. This would allow to go towards a characterization of expected polynomial time computation, thus greatly improving on the existing works on the implicit complexity of probabilistic systems [5,7], which only deals with worst-case execution time. The authors are currently engaged in that.

References

1. Barendregt, H.P.: The Lambda Calculus, Its Syntax and Semantics. Studies in Logic and the Foundations of Mathematics. North-Holland, Amsterdam (1981)
2. Bournez, O., Garnier, F.: Proving positive almost-sure termination. In: Giesl, J. (ed.) RTA 2005. LNCS, vol. 3467, pp. 323–337. Springer, Heidelberg (2005). doi:10.1007/978-3-540-32033-3_24
3. Breuvart, F., Dal Lago, U., Herrou, A.: On probabilistic subrecursion (long version) (2016). http://arxiv.org/abs/1701.04786
4. Crubillé, R., Dal Lago, U.: On probabilistic applicative bisimulation and call-by-value λ-calculi. In: Shao, Z. (ed.) ESOP 2014. LNCS, vol. 8410, pp. 209–228. Springer, Heidelberg (2014). doi:10.1007/978-3-642-54833-8_12
5. Dal Lago, U., Toldin, P.P.: A higher-order characterization of probabilistic polynomial time. Inf. Comput. **241**, 114–141 (2015)
6. Dal Lago, U., Zorzi, M.: Probabilistic operational semantics for the lambda calculus. RAIRO - Theor. Inf. Appl. **46**(3), 413–450 (2012)
7. Dal Lago, U., Zuppiroli, S., Gabbrielli, M.: Probabilistic recursion theory and implicit computational complexity. Sci. Ann. Comp. Sci. **24**(2), 177–216 (2014)
8. De Leeuw, K., Moore, E.F., Shannon, C.E., Shapiro, N.: Computability by probabilistic machines. Automata Stud. **34**, 183–198 (1956)
9. Ehrhard, T., Pagani, M., Tasson, C.: Probabilistic coherence spaces are fully abstract for probabilistic PCF. In: Sewell, P. (ed.) Proceedings of POPL. ACM (2014)
10. Ferrer Fioriti, L.M., Hermanns, H.: Probabilistic termination: soundness, completeness, and compositionality. In: Proceedings of POPL, pp. 489–501 (2015)
11. Gill, J.: Computational complexity of probabilistic turing machines. SIAM J. Comput. **6**(4), 675–695 (1977)
12. Girard, J.-Y., Taylor, P., Lafont, Y.: Proofs and Types. Cambridge University Press, Cambridge (1989)
13. Goodman, N.D., Mansinghka, V.K., Roy, D.M., Bonawitz, K., Tenenbaum, J.B.: Church: a language for generative models. In: UAI, pp. 220–229 (2008)

14. Jung, A., Tix, R.: The troublesome probabilistic powerdomain. Electr. Notes Theor. Comput. Sci. **13**, 70–91 (1998)
15. Kaminski, B.L., Katoen, J.-P.: On the hardness of almost–sure termination. In: Italiano, G.F., Pighizzini, G., Sannella, D.T. (eds.) MFCS 2015. LNCS, vol. 9234, pp. 307–318. Springer, Heidelberg (2015). doi:10.1007/978-3-662-48057-1_24
16. Kozen, D.: Semantics of probabilistic programs. J. Comput. Syst. Sci. **22**(3), 328–350 (1981)
17. Manning, C.D., Schütze, H.: Foundations of Statistical Natural Language Processing, vol. 999. MIT Press, Cambridge (1999)
18. McIver, A., Morgan, C.: Abstraction, Refinement and Proof for Probabilistic Systems. Monographs in Computer Science. Springer, Heidelberg (2005)
19. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, San Mateo (1988)
20. Prince, S.J.D.: Computer Vision: Models, Learning, and Inference. Cambridge University Press, New York (2012)
21. Saheb-Djahromi, N.: Probabilistic LCF. In: Winkowski, J. (ed.) MFCS 1978. LNCS, vol. 64, pp. 442–451. Springer, Heidelberg (1978). doi:10.1007/3-540-08921-7_92
22. Santos, E.S.: Probabilistic turing machines and computability. Proc. Am. Math. Soc. **22**(3), 704–710 (1969)
23. Sørensen, M.H., Urzyczyn, P.: Lectures on the Curry-Howard Isomorphism. Elsevier Science Inc., New York (2006)
24. Statman, R.: The typed lambda-calculus is not elementary recursive. Theor. Comput. Sci. **9**, 73–81 (1979)
25. Staton, S., Yang, H., Heunen, C., Kammar, O., Wood, F.: Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. In: Proceedings of LICS, pp. 525–534 (2016)

Concurrency

A Truly Concurrent Game Model of the Asynchronous π-Calculus

Ken Sakayori(✉) and Takeshi Tsukada

The University of Tokyo, Tokyo, Japan
{sakayori,tsukada}@kb.is.s.u-tokyo.ac.jp

Abstract. In game semantics, a computation is represented by a *play*, which is traditionally a sequence of messages exchanged by a program and an environment. Because of the sequentiality of plays, most game models for concurrent programs are a kind of interleaving semantics. Several frameworks for truly concurrent game models have been proposed, but no model has yet been applied to give a semantics of a complex concurrent calculus such as the π-calculus (with replication).

This paper proposes a truly concurrent version of the HO/N game model in which a play is not a sequence but a directed acyclic graph (DAG) with two kinds edges, justification pointers and causal edges. By using this model, we give the first truly concurrent game semantics for the asynchronous π-calculus. In order to illustrate a possible application, we propose an intersection type system for the asynchronous π-calculus by means of our game model, and discuss when a process can be completely characterised by the intersection type system.

Keywords: HO/N game model · True concurrency · Asynchronous π-calculus

1 Introduction

Game semantics succeeded to give semantics for variety of programming languages such as PCF [1,21] and Idealized Algol [2].

The idea of game semantics has been applied to give models for concurrent calculi such as CSP [23], Idealized Parallel Algol [18] and the asynchronous π-calculus [24]. However, the sequential nature of plays forces these models to be a kind of interleaving semantics; the causalities between events are obfuscated.

Hence it is natural to investigate a concurrent extension of the traditional game models. Several frameworks for concurrent game models have been proposed by several researchers [3,27,29,34], but no model has yet been applied to give a semantics of a complex concurrent calculus such as the π-calculus (with replication), as pointed out in [10]. The goal of this paper is to develop a truly concurrent game model by which the asynchronous π-calculus can be interpreted.

The starting point of our development is an observation by Melliès [27]: in the HO/N innocent game model [21,32], only a part of the sequential information

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 389–406, 2017.
DOI: 10.1007/978-3-662-54458-7_23

Fig. 1. Idea of the desequentialization.

is really relevant. For example, the order of consecutive occurrences of O- and P-moves are indispensable, whereas that of consecutive occurrences of P- and O-moves can be safely forgotten (unless the O-move is justified by the P-move).

Now it is natural to think of a play in which the relevant order information is made explicit. Consider a traditional sequential play on the left side in Fig. 1, where • (resp. ∘) represents an O-move (resp. a P-move) and a pointer is a justification pointer. By making the relevant sequential information explicit, we obtain a representation in the middle in Fig. 1. Then because all the relevant sequential information has been explicitly indicated by edges, we can simply forget the sequential information, resulting in the right representation in Fig. 1. This is our representation of a play that we call a *DAG-based play*.

A DAG-based play generated by this way from a sequential play satisfies a certain property, which reflects the sequential nature of the target language of the innocent game model [21]. In order to model a concurrent calculus, the condition required for DAG-based plays should be weakened. This is the idea that leads us to the definition of plays in this paper.

Following this idea, we develop a DAG-based game model for the asynchronous π-calculus, guided by the sequential game model of Laird [24]. Our model is truly concurrent in the sense that it distinguishes between $a.\bar{b} \mid c.\bar{d}$ and $a.(\bar{b} \mid c.\bar{d}) + c.(a.\bar{b} \mid \bar{d})$. Laird's model can be reconstructed by lining up the nodes of DAG-based plays of our model. We prove the soundness of our model by reducing it to that of Laird's model, using this relationship.

As a possible application of our model, we give an intersection type system based on the relationship between intersection types and game semantics which has been studied in the case of λ-calculus [7,14,37]. Based on a game-semantic consideration, we characterise a class of processes that are completely described by the intersection type system.

Organisation of the paper. Section 2 defines our target language, a variant of the asynchronous π-calculus. In Sect. 3, we define our truly concurrent game model and relate it with sequential game models. A semantics of the π-calculus is given in Sect. 4. Section 5 illustrates a possible application of our game model, giving an intersection type system for a fragment of the π-calculus. Section 6 discusses related work and Sect. 7 concludes the paper.

$$\frac{}{\Gamma \vdash \mathbf{0}; \Sigma} \qquad \frac{\Gamma, \bar{x} : T \vdash P; \Sigma, y : T}{\Gamma \vdash \nu(\bar{x}, y).P; \Sigma} \qquad \frac{}{\Gamma, \bar{x} : \mathbf{ch}[\boldsymbol{S}, \boldsymbol{T}], \bar{\boldsymbol{y}} : \boldsymbol{S} \vdash \bar{x}\langle \bar{\boldsymbol{y}}, \boldsymbol{z} \rangle; \Sigma, \boldsymbol{z} : \boldsymbol{T}}$$

$$\frac{\Gamma \vdash P; \Sigma \quad \Gamma' \vdash Q; \Sigma'}{\Gamma, \Gamma' \vdash P|Q; \Sigma, \Sigma'} \qquad \frac{\Gamma, \bar{\boldsymbol{y}} : \boldsymbol{S} \vdash P; \Sigma, \boldsymbol{z} : \boldsymbol{T}}{\Gamma \vdash x(\bar{\boldsymbol{y}}, \boldsymbol{z}).P; \Sigma, x : \mathbf{ch}[\boldsymbol{S}, \boldsymbol{T}]} \qquad \frac{\Gamma, \bar{\boldsymbol{y}} : \boldsymbol{S} \vdash P; \Sigma, \boldsymbol{z} : \boldsymbol{T}}{\Gamma \vdash !x(\bar{\boldsymbol{y}}, \boldsymbol{z}).P; \Sigma, x : \mathbf{ch}[\boldsymbol{S}, \boldsymbol{T}]}$$

Fig. 2. Typing rules. (Contraction and exchange rules are omitted.)

2 Simply-Typed Asynchronous π-Calculus

We define the target language of the paper: the simply-typed asynchronous polyadic π-calculus with distinction between input and output channels. This is the calculus studied in the previous work of Laird [24], in which he gave an interleaving (or sequential) game model.

We assume countably infinite sets of input names and of output names. Unlike the standard π-calculus in which an input name a is *a priori* connected to the output name $\bar{a}$, we do not assume any relationship between input and output names but a connection is established by ν constructor. This design choice significantly simplifies the denotational semantics.

The *processes* are defined by the following grammar: $P, Q ::= \mathbf{0} \mid \bar{x}\langle \bar{\boldsymbol{y}}, \boldsymbol{z} \rangle \mid x(\bar{\boldsymbol{y}}, \boldsymbol{z}).P \mid P|Q \mid !x(\bar{\boldsymbol{y}}, \boldsymbol{z}).P \mid \nu(\bar{x}, y).P$. Here x (resp. $\bar{x}$) ranges over input (resp. output) names and $\boldsymbol{x}$ (resp. $\bar{\boldsymbol{x}}$) represents a (possibly empty) sequence of input (resp. output) names. Name creation ν creates a pair of input and output names. We abbreviate $\nu(\bar{x}_1, y_1). \ldots .\nu(\bar{x}_n, y_n).P$ as $\nu(\bar{x}_1 \ldots \bar{x}_n, y_1 \ldots y_n).P$.

The structural congruence $\equiv$ is defined as usual. The *one-step reduction relation* $\longrightarrow$ on processes is defined by the following rule:

$$\nu(\bar{\boldsymbol{z}}, \boldsymbol{w}).\nu(\bar{x}, y).(y(\bar{\boldsymbol{a}}, \boldsymbol{b}).P \mid \bar{x}\langle \bar{\boldsymbol{c}}, \boldsymbol{d} \rangle \mid Q) \longrightarrow \nu(\bar{\boldsymbol{z}}, \boldsymbol{w}).\nu(\bar{x}, y).(P\{\bar{\boldsymbol{c}}/\bar{\boldsymbol{a}}, \boldsymbol{d}/\boldsymbol{b}\} \mid Q)$$

It is worth emphasising here that the communication only occurs over names that are bound by ν. The *reduction relation* $\longrightarrow^*$ is the reflexive transitive closure of $(\longrightarrow \cup \equiv)$. We write $P \Downarrow_{\bar{x}}$ if $P \longrightarrow^* \nu(\bar{\boldsymbol{y}}, \boldsymbol{z}).(\bar{x}\langle \bar{\boldsymbol{y}}', \boldsymbol{z}' \rangle \mid Q)$ for some Q, where $\bar{x}$ is free. Note that we can observe only an output action.

We require that processes are well-typed. The syntax of *types* is given by $S, T ::= \mathbf{ch}[S_1 \ldots S_m, T_1 \ldots T_n]$. We write $x : \mathbf{ch}[S_1 \ldots S_m, T_1 \ldots T_n]$ to mean that x is an input name by which one receives m output names and n input names at once. Similarly for $\bar{y} : \mathbf{ch}[S_1 \ldots S_m, T_1 \ldots T_n]$. A sequence $S_1 \ldots S_m$ of types is often written as $\boldsymbol{S}$ and the empty sequence is written as $\_$. The type $\mathbf{ch}[\_, \_]$ is abbreviated as $\mathbf{ch}[]$. An *input type environment* is a finite sequence of type bindings of the form $x : T$ and an *output type environment* is that of the form $\bar{y} : S$. A *type judgement* is of the form $\Gamma \vdash P; \Sigma$, where Γ and Σ are input and output type environments, respectively. Typing rules are listed in Fig. 2.

Remark 1. (1) A calculus with *a priori* connection between an input name x and an output name $\bar{x}$ can be simulated by passing/receiving a pair $(x, \bar{x})$ of input and

output names. Via this translation our game semantics is applicable to a calculus with *a priori* connection because the translation reflects may-testing equivalence. (2) The standard parallel composition, which invokes communications of the two processes, can be expressed as $\nu(\bar{\boldsymbol{a}}\bar{\boldsymbol{b}}, \boldsymbol{ab}).(P|Q|(\boldsymbol{a}' \to \bar{\boldsymbol{a}})|(\boldsymbol{b} \to \bar{\boldsymbol{b}}'))$ where $\boldsymbol{a}$ and $\bar{\boldsymbol{b}}$ are free names in P and Q, and $\boldsymbol{a}' \to \bar{\boldsymbol{a}}$ is a "forwarder", a process forwarding names received from a'_i to $\bar{a}_i$.

3 Concurrent HO/N Game Model

This section introduces a truly concurrent game model in which a play is not a sequence but a directed acyclic graph (DAG). A node of a play is labelled by a move representing an event; an edge represents either a justification pointer or causality. The key is the notion of plays (Sect. 3.2) and of interactions (Sect. 3.3). The other parts are relatively straightforward adaptation of the techniques in the standard HO/N game model (e.g. [21]) or Laird's model [24].

3.1 Arenas

The definition of arenas is (essentially) the same as the definition of arenas in the case of the sequential game model of π-calculus [24]. The differences from the standard definition (e.g. [21]) are (1) all moves are questions, and (2) the owner of moves does not have to alternate.

Definition 1 (Arena). *An* arena *is a triple* $A = (\mathcal{M}_A, \lambda_A, \vdash_A)$*, where* $\mathcal{M}_A$ *is a set of* moves*,* $\lambda_A \colon \mathcal{M}_A \to \{P, O\}$ *is an* ownership function *and* $\vdash_A \subseteq (\{\star\} + \mathcal{M}_A) \times \mathcal{M}_A$ *is an* enabling relation *that satisfies: for every* $m \in \mathcal{M}_A$*, there uniquely exists* $x \in \{\star\} + \mathcal{M}_A$ *such that* $x \vdash_A m$*.*

We say that m is a *P-move* if $\lambda_A(m) = P$; it is an *O-move* if $\lambda_A(m) = O$. Every move represents an output action: a P-move is an output action of the process and an O-move is that of the environment (see a discussion after Definition 4). Let $\lambda_A^{\perp}$ denote the negation of λ_A i.e. $\lambda_A^{\perp}(m) = O$ (resp. $\lambda_A^{\perp}(m) = P$) if $\lambda_A(m) = P$ (resp. $\lambda_A(m) = O$). A move m is *initial* if $\star \vdash_A m$. An arena is *negative* (resp. *positive*) if all initial moves are O-moves (resp. P-moves). In what follows, we shall consider only negative arenas (hence we often use arenas to mean negative arenas). The *empty arena* is defined by $I := (\emptyset, \emptyset, \emptyset)$.

Negative and positive arenas correspond to input and output type environments, respectively. Hence a judgement, which consists of a pair of input and output type environments, should be expressed as a pair of arenas.

Definition 2 (Arena pair). *An* arena pair *is a pair* (A, B) *of (negative) arenas. We write* $\mathcal{M}_{A,B}$ *for* $\mathcal{M}_A + \mathcal{M}_B$*. The ownership function is defined by* $\lambda_{A,B} = [\lambda_A^{\perp}, \lambda_B]$*. The enabling relation* $\vdash_{A,B}$ *is given by:* $m \vdash_{A,B} m'$ *if and only if* $m \vdash_A m'$ *or* $m \vdash_B m'$*. (In particular,* $\star \vdash_{A,B} m$ *iff* $\star \vdash_A m$ *or* $\star \vdash_B m'$*.)*

Note that an arena pair is not a negative arena since it has an initial P-move.

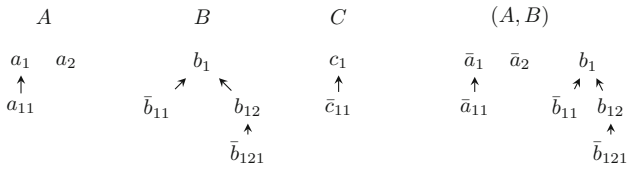


Fig. 3. Examples of arenas and an arena pair.

Example 1. Three (negative) arenas A, B and C are illustrated in Fig. 3, as well as the arena pair (A, B). Those arenas are used in examples in this paper. Nodes are labelled by moves and edges represent the enabling relation. If a name is overlined, the move is a P-move; otherwise it is an O-move. The arena pair (A, B) corresponds to the pair of the output type environment $\Gamma = \bar{a}_1 : \mathbf{ch}[\_, \mathbf{ch}[]], \bar{a}_2 : \mathbf{ch}[]$ and the input type environment $\Sigma = b_1 : \mathbf{ch}[\mathbf{ch}[], \mathbf{ch}[\mathbf{ch}[], \_]]$. (Channel names do not have to coincide with move names.)

3.2 DAG-based Plays

In the standard HO/N game model [21], a play is a sequence of moves equipped with pointers, called *justification pointers*. The justification pointers express the binder-bindee relation and the sequential structure expresses the temporal relation between the events in the sequence (e.g. in the sequence $s_1\, a\, s_2\, b\, s_3$, the event b occurs after a). The causal relation is left implicit (cf. Sect. 3.5).

In the proposed game model, we explicitly describe the causal relation as well as the justification pointers.

Definition 3 (Justified graph). *A* justified graph *over an arena pair* (A, B) *is a tuple* $s = (V_s, l_s, \overset{\frown}{s}, \overset{\rightsquigarrow}{s})$ *where:*

- V_s *is a finite set called the* vertex set
- l_s *is the* vertex labelling*, that is* $l_s \colon V_s \to \mathcal{M}_{A,B}$
- $\overset{\frown}{s} \subseteq V_s \times V_s$ *is the* justification relation
- $\overset{\rightsquigarrow}{s} \subseteq V_s \times V_s$ *is the* causality relation

such that

- $(V_s, \overset{\frown}{s} \cup \overset{\rightsquigarrow}{s})$ *is a DAG i.e. there is no cycle* $v\ (\overset{\rightsquigarrow}{s} \cup \overset{\frown}{s})^+\ v$.
- *If* $l_s(v)$ *is initial, then there is no node* v' *such that* $v \overset{\frown}{s} v'$.
- *If* $l_s(v)$ *is not initial, then there exists a unique node* v' *such that* $v \overset{\frown}{s} v'$. *Furthermore this* v' *satisfies* $l_s(v') \vdash_{A,B} l_s(v)$.

Note that $\overset{\frown}{s}$ *and* $\overset{\rightsquigarrow}{s}$ *do not have to be disjoint. We define* $\vec{s} := (\overset{\frown}{s} \cup \overset{\rightsquigarrow}{s})$. *The set of justified graphs over an arena pair* (A, B) *is denoted by* $J_{A,B}$.

In what follows, we shall identify isomorphic justified graphs.

Given a justified graph s over (A, B), a *P-node* (resp. an *O-node*) is a node $v \in V_s$ whose label is a P-move (resp. an O-move). We write V_s^P for the set of P-nodes and V_s^O for the set of O-nodes (e.g. $V_s^P := \{v \in V_s \mid \lambda_{A,B}(l_s(v)) = P\}$).

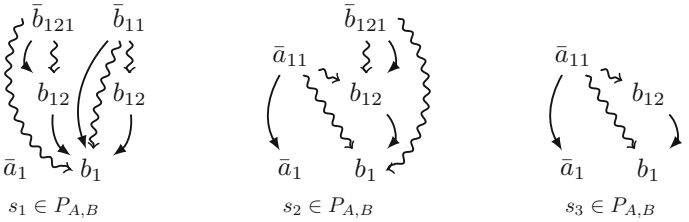


Fig. 4. Examples of plays over the arena pair (A, B) in Fig. 3.

Definition 4 (Play). *Let $s = (V_s, l_s, \curvearrowright_s, \not\rightarrow_s)$ be a justified graph over (A, B). It is a* play *if it satisfies the following conditions:*

(P1) *for every $v, v' \in V_s$, $v \not\rightarrow_s v'$ implies $v \in V_s^P$ and $v' \in V_s^O$,*
(P2) *for every $v_p \in V_s^P$ and $v_o \in V_s^O$, if $v_p \rightarrow_s^+ v_o$, then $v_p \not\rightarrow_s v_o$, and*
(P3) *for every $v_o \in V_s^o$, there exists $v_p \in V_s^P$ such that $v_p \not\rightarrow_s v_o$.*

We write $P_{A,B}$ for the set of plays over (A, B).

Condition (P1) reflects the asynchronous nature of the target language. Recall that a P-move corresponds to an output action of a process and an O-move to an output action of the environment. No P-node should be causally related to P-nodes since an output action of the process cannot cause any other output of the process. Similarly no O-node should be causally related to O-nodes since an output action of the environment cannot cause any other output of the environment (provided that the environment is also described by the asynchronous π-calculus). An output action of a process may cause an output action of the environment; however it is a matter of the environment and a play describes the behaviour of a process, not the environment. Hence $\not\rightarrow_s \subseteq V_s^P \times V_s^O$.

Condition (P2) comes from a purely technical requirement. (We need this condition to establish Lemma 2, as well as a proposition stating the copycat strategy is the identity.)

Condition (P3) is the counterpart of the even-length condition. Here we regard the even-length condition for sequential plays as the requirement that every O-move in the sequence should be responded by a P-move.

Example 2. Figure 4 shows three different plays over the arena pair (A, B) in Fig. 3. The solid arrows represent justification pointers, and squiggly arrows represent causalities. Nodes are labelled by moves and different nodes may be labelled by a same move. Note that plays may have a join point, i.e. a node that is linked to two "incomparable" nodes, like the node labelled by $\bar{a}_{11}$ in s_2.

Remark 2. A play can be seen as a process, e.g. the play s_2 in Fig. 4 corresponds to the process $\nu(\bar{a}_{11}, a_{11}).(b_1(\_, b_{12}).b_{12}(\bar{b}_{121}, \_).(\bar{a}_{11} \mid \bar{b}_{121}) \mid \bar{a}_1\langle \_, a_{11}\rangle)$ (whose type differs from that described by the arena pair). The formal description of the connection to the linear internal π-calculus is left for the future work.

3.3 Strategies and Composition

Strategy. In most variants of sequential game models, a strategy σ is a collection of plays that is *(even-length) prefix closed*: if $sm_Om_P \in \sigma$, then $s \in \sigma$. The set of strategies in our game model is defined by the same way, though the notion of prefix should be adapted to our setting.

Definition 5 (Prefix). *Let $s = (V_s, l_s, \curvearrowright_s, \rightsquigarrow_s)$ be a play. Let $U \subseteq V_s$ be a subset that satisfies* (1) *$v \in U$ and $v \xrightarrow{s} v'$ implies $v' \in U$ and* (2) *for all $v_o \in U^O$ there exists $v_p \in U^P$ such that $v_p \rightsquigarrow_s v_o$. The* prefix $s[U] := (U, l, \curvearrowright, \rightsquigarrow)$ *of s induced by U is the restriction of s to U, i.e.,*

$$l(v) := l_s(v) \quad \curvearrowright := (\curvearrowright_s) \cap (U \times U) \quad \rightsquigarrow := (\rightsquigarrow_s) \cap (U \times U).$$

We write $s' \sqsubseteq s$ if s' is a prefix of s. A prefix of a play is a play.

Example 3. In Fig. 4, the play s_3 is a prefix of s_2 induced by the set of nodes labelled by $m \in \{\bar{a}_1, \bar{a}_{11}, b_1, b_{12}\}$.

Definition 6 (Strategy). *Let (A, B) be an arena pair. A set $\sigma \subseteq P_{A,B}$ of plays over (A, B) is a* strategy *of (A, B), written as $\sigma \colon A \to B$, if it satisfies* prefix-closedness *(S1):*

(S1) *If $s \in \sigma$ and $s' \sqsubseteq s$, then $s' \in \sigma$.*

Composition. The composition of strategies is defined by using the notion of interactions. Since plays are not sequences but graphs, an interaction should also be represented by a graph that we call an *interaction graph.*

Definition 7. *Let (A, B, C) be a triple of arenas. The set $\mathcal{M}_{A,B,C}$ of moves of (A, B, C) is the disjoint union $\mathcal{M}_A + \mathcal{M}_B + \mathcal{M}_C$. The enabling relation $\vdash_{A,B,C}$ is defined by: $x \vdash_{A,B,C} m$ if $x \vdash_X m$ for some $X \in \{A, B, C\}$. The ownership function is defined by: $\lambda_{A,B,C} := [\lambda_A, \lambda_B, \lambda_C]$. The set $J_{A,B,C}$ of* justified graphs of (A, B, C) *is defined by the same way as in Definition 3.*

For $X \in \{A, B, C, (A, B), (B, C), (A, C)\}$, we write V_X for the set of nodes restricted to the component X and V_X^P and V_X^O for the sets of nodes labelled by P-moves and by O-moves in the component X. For example, $v \in V_{A,B}^P$ means either (1) $l_u(v) \in \mathcal{M}_B$ and $\lambda_B(l_u(v)) = P$, or (2) $l_u(v) \in \mathcal{M}_A$ and $\lambda_A(l_u(v)) = O$.

Definition 8 (Restriction). *Let $u = (V, l, \curvearrowright, \rightsquigarrow)$ be a justified graph over (A, B, C) and $X \in \{(A, B), (B, C), (A, C)\}$. The* restriction $u\restriction_X$ *of u to X is defined by $u\restriction_X := (V\restriction_X,\ l\restriction_X,\ \curvearrowright\restriction_X,\ \rightsquigarrow\restriction_X)$, where*

$$V\restriction_X := V_X, \quad l\restriction_X(v) := l(v), \quad \curvearrowright\restriction_X := (\curvearrowright) \cap (V_X \times V_X).$$

The definition of $\rightsquigarrow\restriction_X$ needs some care. If $X \in \{(A, B), (B, C)\}$, then $\rightsquigarrow\restriction_X$ is just the restriction of the original causal relation, i.e. $\rightsquigarrow\restriction_X := \{(v, v') \in V_X^P \times V_X^O \mid v \rightsquigarrow v'\}$ (cf. Condition (P1)). If $X = (A, C)$, then $\rightsquigarrow\restriction_{A,C}$ relates moves linked through the intermediate component B, i.e. $\rightsquigarrow\restriction_{A,C} := \{(v, v') \in V_{A,C}^P \times V_{A,C}^O \mid \exists n \geq 0.\ \exists v_1, \ldots, v_n \in V_B.\ v \rightsquigarrow v_1 \rightsquigarrow \cdots \rightsquigarrow v_n \rightsquigarrow v'\}$.

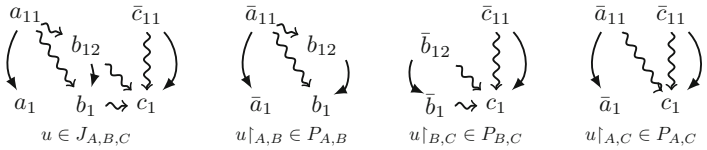


Fig. 5. Example of a justified graph and restrictions.

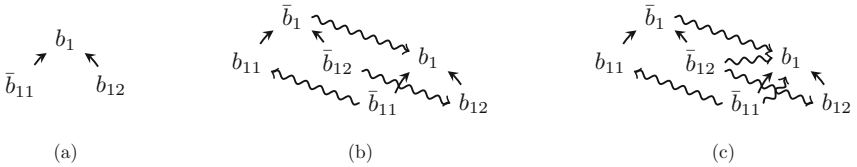


Fig. 6. Construction of a copycat play.

Example 4. Figure 5 shows a justified graph u over the triple (A, B, C) (in Fig. 3) and its restrictions to components (A, B), (B, C) and (A, C).

Note that although $\bar{a}_{11} \not\twoheadrightarrow_u c_1$, we have $\bar{a}_{11} \twoheadrightarrow_u u\restriction_{A,C} c_1$ because $\bar{a}_{11} \twoheadrightarrow_u b_1 \twoheadrightarrow_u c_1$.

Definition 9 (Interaction graph). *Let $u \in J_{A,B,C}$ be a justified graph over (A, B, C) and V be the set of nodes of u. We say that u is an* interaction graph *if it satisfies the following conditions.*

(I1) *If $v \twoheadrightarrow_u v'$, then $(v, v') \in V_X^P \times V_X^O$ for some $X \in \{(A, B), (B, C)\}$.*
(I2) *Both $u\restriction_{A,B}$ and $u\restriction_{B,C}$ are plays.*

Condition (I1) is a variant of the switching condition. *The set of interaction graphs over (A, B, C) is denoted as* $\mathrm{Int}(A, B, C)$.

In fact u in Example 4 is an interaction graph.

Definition 10 (Composition). *Let $\sigma\colon A \to B$ and $\tau\colon B \to C$ be strategies. The* composition *of σ and τ is defined by*

$$\tau \circ \sigma := \{u\restriction_{A,C} \mid u \in \mathrm{Int}(A, B, C),\ \ u\restriction_{A,B} \in \sigma,\ \ u\restriction_{B,C} \in \tau\}.$$

Note that the definition of composition is applicable to sets of plays that are not necessarily strategies. By abuse of notation, we shall write $\tau \circ \sigma$ even if σ and τ are not strategies but just sets of plays.

Theorem 1. *The composite of strategies is a strategy. The composition is associative.*

Category. We define the category $\mathcal{P}$ of negative arenas and strategies: an object of $\mathcal{P}$ is a (negative) arena and a morphism from A to B is a strategy $\sigma\colon A \to B$. The composite of $\sigma\colon A \to B$ and $\tau\colon B \to C$ is given by the composition $\tau \circ \sigma$ of strategies defined above. Given an arena A, the identity morphism $\mathrm{id}_A\colon A \to A$ is the "copycat strategy": when the environment makes a move m in one component, then it responds by making a copy of m in the other component. It is the set of *copycat plays*, whose construction is illustrated in Fig. 6: (*a*) take a "justified graph without causality" of the arena (in this example, the arena is B in Fig. 3); (*b*) make positive and negative copies and connect the corresponding nodes by a causal edge $\rightsquigarrow$ in the appropriate direction; and (*c*) add causal edges so as to satisfy Condition (P2), resulting in a play over (B, B).

3.4 Distributive-Closed Freyd Category

In this section, we define the categorical structures of $\mathcal{P}$, which is used in Sect. 4 to give an interpretation of the π-calculus. A category with the structures below is called a *distributive-closed Freyd category* [24]. The definitions in this section are adapted from the interleaving game model for the π-calculus [24].

Monoidal product. Let $A = (\mathcal{M}_A, \lambda_A, \vdash_A)$ and $B = (\mathcal{M}_B, \lambda_B, \vdash_B)$ be arenas. The arena $A \odot B$ is defined as $(\mathcal{M}_A + \mathcal{M}_B, [\lambda_A, \lambda_B], \vdash_{A,B})$, where $\vdash_{A,B}$ is the enabling relation defined in Definition 2. Given strategies $\sigma\colon A \to B$ and $\tau\colon C \to D$, the strategy $\sigma \odot \tau\colon A \odot C \to B \odot D$ is defined by the juxtaposition of plays in σ and τ, namely $\sigma \odot \tau := \{s \uplus t \mid s \in \sigma, t \in \tau\}$ where $s \uplus t$ is the juxtaposition of plays. Then the triple $(\mathcal{P}, \odot, I)$ is a symmetrical monoidal category.

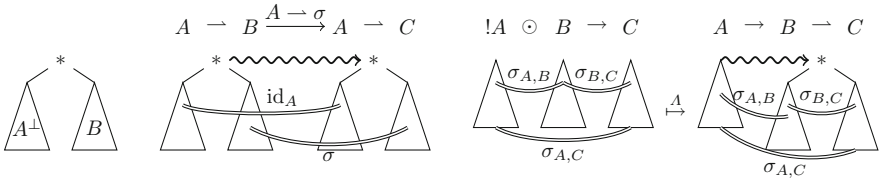
Closed Freyd structure. An input prefixing $a(\bar{\boldsymbol{x}}, \boldsymbol{y}).P$ should be interpreted by using a kind of closed structure (intuitively because the input prefix bounds variables in P like λ-abstraction). Laird [24] used *closed Freyd categories* [33].

A Freyd category consists of a symmetric (pre)monoidal category $\mathcal{P}$, a cartesian category $\mathcal{A}$ and an identity-on-object strict (pre)monoidal functor $!\colon \mathcal{A} \to \mathcal{P}$. Intuitively $\mathcal{P}$ is that of types and "terms" whereas $\mathcal{A}$ is the category of types and "values"; the functor ! gives us a way to regard a "value" as a "term". In our context, "terms" are processes and "values" are processes of the form $\sum_i a_i(\bar{\boldsymbol{x}}_i, \boldsymbol{y}_i).P_i$, where P_i has no free input channel except for those in $\boldsymbol{y}_i$.

We define the game-semantic counterpart of the processes of the this form.

Definition 11 (Well-opened play, strategy). *A play s is* well-opened *if it contains precisely one initial O-node v_0 to which all other nodes are connected (i.e. $v \stackrel{s}{\Rightarrow}{}^{*} v_0$ for every $v \in V_s$). We write $W_{A,B}$ for the set of well-opened plays over (A, B). A* well-opened strategy *from arena A to arena B, written as $\sigma\colon A \stackrel{\bullet}{\to} B$, is a set σ of well-opened plays that is prefix-closed (S1).*

Then we define an operator !, a mapping from well-opened strategies to strategies and the composition of well-opened strategies by using !.



(a) $A \multimap B$. (b) $A \multimap \sigma$. (c) $\Lambda(\sigma)$.

Fig. 7. The action of $A \multimap (-)$ and Λ.

Definition 12. *Let $\sigma\colon A \xrightarrow{\bullet} B$ be a well-opened strategy. The strategy $!\sigma\colon !A \to B$ is defined by $!\sigma := \{s_1 \uplus \cdots \uplus s_n \mid n \geq 0,\ \forall i \leq n.\, s_i \in \sigma\}$ where $s_1 \uplus \cdots \uplus s_n$ is the juxtaposition of plays $s_1, \ldots, s_n$.*

Definition 13 (Composition of well-opened strategies). *Let $\sigma\colon A \xrightarrow{\bullet} B$ and $\tau\colon B \xrightarrow{\bullet} C$ be well-opened strategies. We define $\tau \circ_{\mathcal{A}} \sigma := \tau \circ !\sigma$.*

Lemma 1. *The composite of well-opened strategies with respect to $\circ_{\mathcal{A}}$ is a well-opened strategy. The composition $\circ_{\mathcal{A}}$ of well-opened strategies is associative.*

The *category $\mathcal{A}$ of negative arenas and well-opened strategies* is defined by the following data: an object is a negative arena, a morphism from A to B is a well-opened strategy $\sigma\colon A \xrightarrow{\bullet} B$, the composition is given by $\circ_{\mathcal{A}}$. The identity morphism is $\mathrm{id}_A \cap W_{A,A}$, where id_A is the copycat strategy. The category $\mathcal{A}$ is cartesian: the cartesian product of A and B is $A \odot B$.

By defining $!A := A$ for objects, the operation $!$ becomes a functor $!\colon \mathcal{A} \to \mathcal{P}$. This is identity on objects and strict symmetric monoidal functor and thus $(\mathcal{A}, \mathcal{P}, !)$ is a Freyd category.

Lemma 2. *The Freyd category $(\mathcal{A}, \mathcal{P}, !)$ is closed, i.e. for every arena A, the functor $!(-) \odot A\colon \mathcal{A} \to \mathcal{P}$ has the right-adjoint $A \multimap (-)\colon \mathcal{P} \to \mathcal{A}$.*

The action of $A \multimap (-)$ on objects and on morphisms is illustrated in Fig. 7. We write Λ for the bijective map $\mathcal{P}(!A \odot B, C) \to \mathcal{A}(A, B \multimap C)$ and $\mathbf{app}_{A,B}\colon !(A \multimap B) \odot A \to B$ for the counit. The bijection $\mathcal{P}(!A \odot B, C) \cong \mathcal{A}(A, B \multimap C)$ induced by the adjunction intuitively corresponds to the following bijection of the π-calculus processes: $\bar{\boldsymbol{x}} : \boldsymbol{S}, \bar{\boldsymbol{y}} : \boldsymbol{T} \vdash P; \boldsymbol{z} : \boldsymbol{U} \quad \longleftrightarrow \quad \bar{\boldsymbol{x}} : \boldsymbol{S} \vdash a(\bar{\boldsymbol{y}}, \boldsymbol{z}).P; a : \mathbf{ch}[\boldsymbol{T}, \boldsymbol{U}]$.

Distributive law. The process obtained by (the π-term representation of) the above adjunction has the input prefix $a(\bar{\boldsymbol{y}}, \boldsymbol{z})$ as expected but it has only one free input channel. We use the *distributive law* of the distributive-closed Freyd category to model a process with multiple free input channel. By using the syntax of the π-calculus, the distribution law can be seen as the following map:

$$\bar{\boldsymbol{x}}\colon \boldsymbol{S} \vdash a(\bar{\boldsymbol{y}}, \boldsymbol{z}\boldsymbol{z}').P;\ a\colon \mathbf{ch}[\boldsymbol{T}, \boldsymbol{U}\boldsymbol{U}'] \ \longrightarrow\ \bar{\boldsymbol{x}}\colon \boldsymbol{S} \vdash a(\bar{\boldsymbol{y}}, \boldsymbol{z}).P;\ a\colon \mathbf{ch}[\boldsymbol{T}, \boldsymbol{U}], \boldsymbol{z}'\colon \boldsymbol{U}'.$$

Definition 14 (Distributive-closed Freyd category [24]). *A closed Freyd category* $!\colon \mathcal{A} \to \mathcal{P}$ *is* distributive-closed *if there is a family of morphisms* $\varrho_A\colon !(A \multimap (B \odot C)) \longrightarrow B \odot !(A \multimap C)$ *in* $\mathcal{P}$*, natural in* B *and* C *which makes certain diagrams commute.*

Theorem 2. *The game model* $!\colon \mathcal{A} \to \mathcal{P}$ *is distributive-closed.*

Trace. The operator $\nu(\bar{x}, y).P$ is interpreted as a trace operator. We define $Tr^B_{A,C}(f) := \mathbf{app}_{B,C} \circ \mathbf{symm}_{B,B\multimap C} \circ \varrho_{B,B,C} \circ !\Lambda(\mathbf{symm}_{B,C} \circ f)$, given a morphism $f\colon A\odot B \to C\odot B$ in $\mathcal{P}$. Then Tr is the trace operator for the symmetrical monoidal category $\mathcal{P}$ [24].

Additional structures. Some additional structures are required to interpret the π-calculus: the *minimum strategy* $\bot_{A,B}$ (with respect to the set-inclusion), the *diagonal* $\Delta_A\colon A \xrightarrow{\bullet} A \odot A$, the *codiagonal* $\nabla_A\colon A \odot A \xrightarrow{\bullet} A$ (defined by $\nabla_A := \pi_1 \cup \pi_2$ where $\pi_i\colon A\odot A \xrightarrow{\bullet} A$ is the projection), and the *dereliction* $der_A\colon A \to A$ (defined as $\mathrm{id}_A \cap W_{A,A}$).

3.5 Relation to Sequential Game Models

Laird's interleaving game model. Our model can be seen as a truly concurrent version of the interleaving game model $\mathcal{P}_{\mathrm{L}}$ of Laird [24]. The idea is to relate a (concurrent) play $s = (V_s, l_s, \curvearrowright_s, \rightarrow_s)$ to an interleaving play by lining up the nodes in V_s in such a way that if $v_1 \rightarrow_s v_2$, then v_2 appears before v_1. We write $|s|$ for the set of sequential plays obtained by this way.

Example 5. Let s_2 be the play in Fig. 4. Then $|s_2|$ is given as:

$$\left\{ \bar{a}_1 b_1 b_{12} \bar{a}_{12} \bar{b}_{121},\ \bar{a}_1 b_1 b_{12} \bar{b}_{121} \bar{a}_{12},\ b_1 \bar{a}_1 b_{12} \bar{a}_{12} \bar{b}_{121},\ b_1 \bar{a}_1 b_{12} \bar{b}_{121} \bar{a}_{12},\ b_1 b_{12} \bar{a}_1 \bar{a}_{12} \bar{b}_{121},\ b_1 b_{12} \bar{a}_1 \bar{b}_{121} \bar{a}_{12},\ b_1 b_{12} \bar{b}_{121} \bar{a}_1 \bar{a}_{12} \right\}$$

Theorem 3. $|-|$ *induces an identity-on-object functor from* $\mathcal{P}$ *to* $\mathcal{P}_{\mathrm{L}}$*, which preserves the structure of distributed-closed Freyd categories (and the additional structures). Furthermore* $|\sigma|$ *is the minimum strategy if and only if so is* σ*.*

Sequential HO/N game model. The standard sequential HO/N game model [21] is a subcategory of our concurrent model. Since our game model only have question moves, we compare our model with the HO/N game model without *answer* (and thus without *well-bracketing*).

An arena A is *alternating* if $m \vdash_A m'$ implies $\lambda_A(m) = \lambda_A^{\perp}(m')$. Let $\mathcal{G}$ be the category of negative alternating arenas and innocent strategies (we omit the definition, which is standard). We write $\ulcorner \hat{s} \urcorner$ for the *P-view* [21] of the sequential play $\hat{s}$. Given a sequential play $\hat{s} = m_1 \dots m_n$, a DAG-based play is given by

$$\|\hat{s}\| := (V_{\hat{s}}, l_{\hat{s}},\ \{(i,j) \mid \rho_{\hat{s}}(i) = j\},\ \{(i,j) \in V^P_{\hat{s}} \times V^O_{\hat{s}} \mid m_j \in \ulcorner m_1 \dots m_i \urcorner\})$$

where $V_{\hat{s}} := \{1, \ldots, n\}$, $l_{\hat{s}}(i) := m_i$ and $\rho_{\hat{s}}$ is the partial function describing the justification pointer. Note that the occurrence m_i of a P-move is causally related to an occurrence m_j of an O-move if and only if m_j appears in the P-view of m_i. This map is naturally extended to strategies, namely $\|\hat{\sigma}\| := \{\|\hat{s}\| \mid \hat{s} \in \hat{\sigma}\}$.

Theorem 4. *$\|-\|$ induces a faithful functor from $\mathcal{G}$ to $\mathcal{P}$.*

Remark 3. It is natural to ask if one can give a similar map from Laird's interleaving model. The answer seems negative: all maps that we have checked are not functorial. See [8] for a related result.

4 Game Semantics of the π-calculus

We give an interpretation of the π-calculus, following the result of Laird [24] applicable to every distributive-closed Freyd category with additional structures.

A type and a type environment are interpreted as objects of $\mathcal{P}$. The interpretation of a type $\mathbf{ch}[\boldsymbol{S}, \boldsymbol{T}]$ and a sequence $\boldsymbol{S}$ of types are defined by:

$$[\![\mathbf{ch}[\boldsymbol{S}, \boldsymbol{T}]]\!] := [\![\boldsymbol{S}]\!] \multimap [\![\boldsymbol{T}]\!] \quad [\![S_1 \ldots S_n]\!] := [\![S_1]\!] \odot \cdots \odot [\![S_n]\!] \quad [\![\_]\!] := I.$$

The interpretation of an input type environment is given by the tensor product of elements, e.g. $[\![x_1 : S_1, \ldots, x_n : S_n]\!] := [\![S_1]\!] \odot \cdots \odot [\![S_n]\!]$.

A process $\Gamma \vdash P; \Sigma$ is interpreted as a morphism $[\![P]\!] \colon [\![\Gamma]\!] \to [\![\Sigma]\!]$ in $\mathcal{P}$. The interpretation is defined by induction on the type derivations. The rules are listed in Fig. 8.

$$\begin{aligned}
[\![\Gamma \vdash \mathbf{0}; \Sigma]\!] &= \bot_{[\![\Gamma]\!],[\![\Sigma]\!]} \\
[\![\Gamma, \Gamma' \vdash P|Q; \Sigma, \Sigma']\!] &= [\![P]\!] \odot [\![Q]\!] \\
[\![\Gamma, \bar{x} : \mathbf{ch}[\boldsymbol{S}, \boldsymbol{T}], \bar{\boldsymbol{y}} : \boldsymbol{S} \vdash \bar{x}\langle \bar{\boldsymbol{y}}, \boldsymbol{z}\rangle; \Sigma, \boldsymbol{z} : \boldsymbol{T}]\!] &= \bot_{[\![\Gamma]\!],[\![\Sigma]\!]} \odot \mathbf{app}_{[\![\boldsymbol{S}]\!],[\![\boldsymbol{T}]\!]} \\
[\![\Gamma \vdash x(\bar{\boldsymbol{y}}, \boldsymbol{z}).P; \Sigma, x : \mathbf{ch}[\boldsymbol{S}, \boldsymbol{T}]]\!] &= (\mathrm{id}_{[\![\Sigma]\!]} \odot der_{[\![(\boldsymbol{S},\boldsymbol{T})]\!]}) \circ [\![!x(\bar{\boldsymbol{y}}, \boldsymbol{z}).P]\!] \\
[\![\Gamma \vdash !x(\bar{\boldsymbol{y}}, \boldsymbol{z}).P; \Sigma, x : \mathbf{ch}[\boldsymbol{S}, \boldsymbol{T}]]\!] &= \varrho_{[\![\boldsymbol{S}]\!],[\![\Sigma]\!],[\![\boldsymbol{T}]\!]} \circ !\Lambda([\![P]\!]) \\
[\![\Gamma \vdash \nu(\bar{x}, y).P; \Sigma]\!] &= Tr^{[\![T]\!]}_{[\![\Gamma]\!],[\![\Sigma]\!]}([\![P]\!])
\end{aligned}$$

Fig. 8. Interpretation of processes. (Contraction and exchange rules are omitted.)

The distributive-closed Freyd structure together with additional structures (of Δ, ∇, $\bot$, *der*) gives a (weak) soundness result with respect to the reduction.

Theorem 5. *Let $\Gamma \vdash P; \Sigma$ and $\Gamma \vdash Q; \Sigma$ be processes of the same type.*

1. *If $P \equiv Q$, then $[\![\Gamma \vdash P; \Sigma]\!] = [\![\Gamma \vdash Q; \Sigma]\!]$.*
2. *If $P \longrightarrow Q$, then $[\![\Gamma \vdash P; \Sigma]\!] \supseteq [\![\Gamma \vdash Q; \Sigma]\!]$.*

The relationship to Laird's model (Theorem 3) gives a finer result, which does not follow from the general theory of the distributive-closed Freyd categories.

Lemma 3. (Adequacy). *$P \Downarrow_{\bar{x}}$ iff $[\![P]\!] \neq \bot$ for every process $\bar{x} : \mathbf{ch}[] \vdash P; \_$.*

Proof. Because of Theorem 3, we have $|[\![P]\!]| = [\![P]\!]_L$, where $[\![P]\!]_L$ is the interpretation of the process in Laird's game model [24]. Laird [24] shows that $P \Downarrow_{\bar{x}}$ if and only if $[\![P]\!]_L \neq \bot$. Since $|-|$ preserves $\bot$, we obtain the claim. □

Lemma 3 and monotonicity of the interpretation lead to the next theorem.

Theorem 6. *Let $\bar{x}$ be a testing name that does not occur in Γ. If $[\![\Gamma \vdash P; \Sigma]\!] \subseteq [\![\Gamma \vdash Q; \Sigma]\!]$, then $C[P] \Downarrow_{\bar{x}}$ implies $C[Q] \Downarrow_{\bar{x}}$ for all context $C[\,]$.*

Unlike Laird's model [24], our model is not complete since our model is truly concurrent. For example, $[\![a().\bar{b}\langle\rangle \mid c().\bar{d}\langle\rangle]\!] \neq [\![\nu(\bar{x}, x).(\bar{x}\langle\rangle \mid x().a().(\bar{b}\langle\rangle \mid c().\bar{d}\langle\rangle) \mid x().c().(a().\bar{b}\langle\rangle \mid \bar{d}\langle\rangle))]\!]$ in our model, whereas they are testing equivalent.

5 Discussion: Relationally-Describable Process

Using our game model, we study the relational interpretations of process in the form of intersection type system that describes the behaviour of processes. The intersection type system is a fully abstract model for *a class of* process which we characterise with the help of "interaction graph".

The syntax of *types* and *intersections* are defined by the following grammar:

$$\varphi, \psi ::= \mathbf{ch}[\xi_1 \dots \xi_n, \zeta_1 \dots \zeta_k] \qquad \xi, \zeta ::= \langle \varphi_1, \dots, \varphi_n \rangle$$

where $\langle \cdots \rangle$ is a finite multiset defined by an enumeration of the elements. A *type environment* is a sequence of type bindings of the form $x{:}\xi$ (or $\bar{y}{:}\zeta$). Given intersections $\xi = \langle \varphi_1, \dots, \varphi_n \rangle$ and $\zeta = \langle \psi_1, \dots, \psi_k \rangle$, we write $\xi \wedge \zeta$ for $\langle \varphi_1, \dots, \varphi_n, \psi_1, \dots, \psi_k \rangle$. This operation is extended to type environments by pointwise application. The typing rules are listed below (some rules are omitted):

$$\frac{}{\emptyset \vdash \mathbf{0};\ \emptyset} \qquad \frac{\Xi \vdash P;\ \Theta \qquad \Xi' \vdash P';\ \Theta'}{\Xi, \Xi' \vdash P|P';\ \Theta, \Theta'} \qquad \frac{}{\bar{x} : \mathbf{ch}[\boldsymbol{\xi}, \boldsymbol{\zeta}], \bar{\boldsymbol{y}} : \boldsymbol{\xi} \vdash \bar{x}\langle \bar{\boldsymbol{y}}, \boldsymbol{z} \rangle;\ \Theta, \boldsymbol{z} : \boldsymbol{\zeta}}$$

$$\frac{\Xi, \bar{\boldsymbol{y}}{:}\boldsymbol{\xi} \vdash x(\bar{\boldsymbol{y}}, \boldsymbol{z}).P;\ \Theta, \boldsymbol{z}{:}\boldsymbol{\zeta}}{\Xi \vdash x(\bar{\boldsymbol{y}}, \boldsymbol{z}).P;\ \Theta, x{:}\mathbf{ch}[\boldsymbol{\xi}, \boldsymbol{\zeta}]} \qquad \frac{\forall i \in I.\ \Xi_i \vdash x(\bar{\boldsymbol{y}}, \boldsymbol{z}).P;\ \Theta_i}{\bigwedge_{i \in I} \Xi_i \vdash !x(\bar{\boldsymbol{y}}, \boldsymbol{z}).P;\ \bigwedge_{i \in I} \Theta_i} \qquad \frac{\Xi, \bar{x}{:}\xi \vdash P;\ \Theta, y{:}\xi}{\Xi \vdash \nu(\bar{x}, y).P;\ \Theta}$$

This type system is inspired by the correspondence between intersection type systems and the operation called *time forgetting map* [4], which is an operation that forgets the temporal structure of plays, in sequential game models (see, e.g., [37]). Time forgetting map is the operation that forgets the causal relation in the case of our concurrent game model.

Completeness of the type system holds for every process, but soundness does not; the reason is explained by a game-semantic consideration. We would thus like to find a class for which the relational interpretation is sound.

Let $s \in P_{A,B}$ and $t \in P_{B,C}$ be plays. We say that s and t are *composable* if $s{\restriction}_B$ coincides with $t{\restriction}_B$ except for the causal relations. Then it would be natural

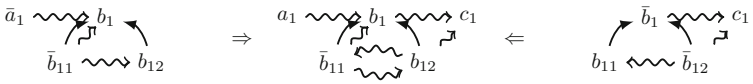


Fig. 9. Composable plays with a cycle.

to think of an "interaction graph" by composing them (see Fig. 9). Unfortunately the resulting "interaction graph" may not be acyclic and hence not be an interaction graph; in this case we say that the pair (s, t) contains a cycle.

This notion of cycle can be extended to strategies and processes. The composition of strategies $\tau \circ \sigma$ is *cycle-free* if every pair of composable plays $s \in \sigma$ and $t \in \tau$ is cycle-free. A process P is *relationally-describable* if every composition in the definition of $[\![P]\!]$ is cycle-free.

Theorem 7. *Let* $\Gamma \vdash P;\ \Sigma$ *be a relationally-describable process and let* $\tilde{x} \in \mathrm{dom}(\Gamma)$. *Then* $P \Downarrow_{\tilde{x}}$ *if and only if* $\tilde{x} : \mathbf{ch}[\boldsymbol{\xi}, \boldsymbol{\zeta}] \vdash P;\ \emptyset$ *for some* $\boldsymbol{\xi}$ *and* $\boldsymbol{\zeta}$.

This is because the operation of forgetting the causal relation commutes with cycle-free composition. Note that the notion of cycle is stronger than deadlock: $\nu(\bar{a}\bar{b}, ab).(a_1.\bar{b}_2 | b_3.\bar{a}_4 | \bar{a}_5)$ (subscripts are used to distinguish occurrences) is *not* relationally-describable because connecting a_1 to $\bar{a}_4$ and b_3 to $\bar{b}_2$ creates a cycle.

Restricting the form of processes by focusing on cycles is a reminiscent of the correctness criterion for MLL proof nets. The formal relationship between our notion of cycle in an interaction graph and the correctness criterion, and the connection between cycle (in our sense) and the type system, which gives a typed π-calculus corresponding to polarised proof-nets satisfying the correctness criterion, proposed by Honda and Laurent [19] are worth investigating.

6 Related Work

Melliès [27] studied HO/N innocent strategies from a truly concurrent point of view. Among others, he introduced the notions of *alternating homotopy* and *diagrammatic innocence*, which influence to this work. These ideas were subsequently developed by Melliès and Mimram [29,30], who introduced *asynchronous games*. They focused on the fact that some moves of a play in an innocent strategy can be exchanged, and studied games whose rules explicitly describe which moves should be commutable. Our game model is also inspired by [27] (and [28]) but we focused on a different aspect of the alternating homotopy, that is, the fact that the connection between a successive pair of O- and P-moves (in HO/N innocent strategies) are quite tight (see also [25,36]); in our game model, a strategy explicitly describes indispensable connections $\rightsquigarrow$ between events. Because of these differences, their game model differs from ours; indeed our strategy is not necessarily *positional*. Nevertheless those models seems closely related; for example, it seems worth investigating the connection between *scheduled strategies* [30] and cycle-free composition.

A related approach using a map of *event structures* has been proposed by Rideau and Winskel [34] and extensively studied recently [9,10]. In this game model, a strategy is a map from an event structure describing the internal causal relation to another event structure expressing the observable events. We think that their model should be closely related to the (pre)sheaf version [36] of our game model, although we have not established any formal relationship yet.

From a technical point of view, an important difference between above models and our model is the way to deal with duplication of moves. Our model uses HO/N-style justification pointers, whereas the above models use the idea of *thread indexing* [10,26] in the style of AJM game model [1]. Both approaches have advantages and disadvantages (for example, an advantage of the HO/N-style is that a morphism is a strategy, not an equivalence class of strategies modulo reindexing). Hence we think that it is good to have a truly concurrent model using justification pointers.

Laird [24] briefly discussed an idea of a truly concurrent version of his interleaving game model, introducing the notion of *justified pomset*. His idea is very closed to ours; indeed a play s in our game model can be seen as a pomset $(V_s, \overrightarrow{s}^*)$ ordered by (reflexive transitive closure of) the adjacent relation $\overrightarrow{s}^*$.

A DAG-based reformulation of the HO/N game model is a reminiscent of *L-nets* [13,17]. The conditions required for L-nets are essentially the same as those we require for plays, though L-nets corresponds to strategies, not to plays. An interpretation of the π-calculus using *differential nets* [15] seems to be relevant to our development.

The game-semantics study of this paper has many parallels to the syntactic study of the π-calculus. The relationship between the HO/N game model for PCF [21] and the π-calculus has originally been studied by Hyland and Ong themselves [20], who gave a translation from PCF terms to processes of the π-calculus based on the idea of their game model. The π-terms representing sequential functional computation can be characterised by a simple type system proposed by Berger, Honda and Yoshida [5], which lead to the type system of [19]. We conjecture that processes typed by the simple type system of [5,19] is related to relationally-describable processes. Boreale [6] gave an encoding from the asynchronous π-calculus to the *internal* π *-calculus* [35]. Our game model can be seen as a variant of the encoding by regarding the plays as the processes of the *linear* internal π-calculus, in which each name must be used exactly once.

There are some pieces of work based on the techniques other than games but related to this work, such as event structure semantics of several variants of the π-calculus by Crafa, Varacca and Yoshida [11,12] and Varacca and Yoshida [38], and a data-flow semantics by Jagadeesan and Jagadeesan [22].

7 Conclusion and Future Work

We have developed a truly concurrent version of the HO/N game model [21,32], in which a computation is represented by a DAG of messages instead of a sequence. The resulting game model has the categorical structure needed to

interpret the asynchronous π-calculus proposed by Laird [24]. By using the connection between our model and Laird's model [24], we have proved soundness of the interpretation of the processes in our concurrent game model. This is the first truly concurrent game semantics for the π-calculus.

We have several topics left for future work:

- Formal description of the connection between plays and processes mentioned in Remark 2. By this connection, our game semantics can be seen as an approximation of the processes of the π-calculus by a linear π-calculus, which is a reminiscent of the *Taylor expansion* of the λ-calculus [16] (see also [37]).
- Development of the (pre)sheaf version of the game model [36], which would be related to the game model based on [34].
- Development of a model of the synchronous π-calculus. This requires us to deal with causal edges from O-moves and/or to P-moves. To simply relax the requirements for $\rightsquigarrow$ does not seem to work: for example, the copycat strategy of this paper is no longer the identity in the relaxed version.
- Development of a model of the π-calculus with the matching primitive. We expect that a nominal game model [31] would be useful for this purpose.

Acknowledgements. We would like to thank Naoki Kobayashi and anonymous referees for useful comments. This work is partially supported by JSPS Kakenhi Grant Number 15H05706 and JSPS Kakenhi Grant Number 16K16004.

References

1. Abramsky, S., Jagadeesan, R., Malacaria, P.: Full abstraction for PCF. Inf. Comput. **163**(2), 409–470 (2000)
2. Abramsky, S., McCusker, G.: Linearity, sharing and state: a fully abstract game semantics for idealized algol with active expressions. Electr. Notes Theor. Comput. Sci. **3**, 2–14 (1996)
3. Abramsky, S., Melliès, P.-A.: Concurrent games and full completeness. In: 14th Annual IEEE Symposium on Logic in Computer Science, pp. 431–442 (1999)
4. Baillot, P., Danos, V., Ehrhard, T., Regnier, L.: Timeless games. In: 11th International Workshop on Computer Science Logic, pp. 56–77 (1997)
5. Berger, M., Honda, K., Yoshida, N.: Sequentiality and the pi-calculus. In: TLCA, pp. 29–45 (2001)
6. Boreale, M.: On the expressiveness of internal mobility in name-passing calculi. Theor. Comput. Sci. **195**(2), 205–226 (1998)
7. Boudes, P.: Thick subtrees, games and experiments. In: Curien, P.-L. (ed.) TLCA 2009. LNCS, vol. 5608, pp. 65–79. Springer, Heidelberg (2009). doi:10.1007/978-3-642-02273-9_7
8. Castellan, S., Clairambault, P.: Causality vs. interleavings in concurrent game semantics. In: 27th International Conference on Concurrency Theory, CONCUR 2016, pp. 32:1–32:14 (2016)
9. Castellan, S., Clairambault, P., Winskel, G.: Symmetry in concurrent games. In: Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS 2014, pp. 28:1–28:10 (2014)

10. Castellan, S., Clairambault, P., Winskel, G.: The parallel intensionally fully abstract games model of PCF. In: 30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, pp. 232–243 (2015)
11. Crafa, S., Varacca, D., Yoshida, N.: Compositional event structure semantics for the internal *pi*-calculus. In: CONCUR 2007 - Concurrency Theory, 18th International Conference, CONCUR 2007, pp. 317–332 (2007)
12. Crafa, S., Varacca, D., Yoshida, N.: Event structure semantics of parallel extrusion in the pi-calculus. In: Foundations of Software Science and Computational Structures - 15th International Conference, FOSSACS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, pp. 225–239 (2012)
13. Curien, P.-L., Faggian, C.: An approach to innocent strategies as graphs. Inf. Comput. **214**, 119–155 (2012)
14. Di Gianantonio, P., Lenisa, M.: Innocent game semantics via intersection type assignment systems. In: Computer Science Logic 2013, CSL 2013, pp. 231–247 (2013)
15. Ehrhard, T., Laurent, O.: Interpreting a finitary pi-calculus in differential interaction nets. Inf. Comput. **208**(6), 606–633 (2010)
16. Ehrhard, T., Regnier, L.: Uniformity and the taylor expansion of ordinary lambda-terms. Theor. Comput. Sci. **403**(2–3), 347–372 (2008)
17. Faggian, C., Maurel, F.: Ludics nets, a game model of concurrent interaction. In: 20th IEEE Symposium on Logic in Computer Science (LICS 2005), pp. 376–385 (2005)
18. Ghica, D.R., Murawski, A.S.: Angelic semantics of fine-grained concurrency. Ann. Pure Appl. Logic **151**(2–3), 89–114 (2008)
19. Honda, K., Laurent, O.: An exact correspondence between a typed pi-calculus and polarised proof-nets. Theor. Comput. Sci. **411**(22–24), 2223–2238 (2010)
20. Hyland, J.M.E., Ong, C.-H.L.: Pi-calculus, dialogue games and PCF. In: Proceedings of the Seventh International Conference on Functional Programming Languages and Computer Architecture, FPCA 1995, pp. 96–107 (1995)
21. Hyland, J.M.E., Ong, C.-H.L.: On full abstraction for PCF: I, II, and III. Inf. Comput. **163**(2), 285–408 (2000)
22. Jategaonkar Jagadeesan, L., Jagadeesan, R.: Causality and true concurrency: a data-flow analysis of the pi-calculus. In: Alagar, V.S., Nivat, M. (eds.) AMAST 1995. LNCS, vol. 936, pp. 277–291. Springer, Heidelberg (1995). doi:10.1007/3-540-60043-4_59
23. Laird, J.: A game semantics of idealized CSP. Electr. Notes Theor. Comput. Sci. **45**, 232–257 (2001)
24. Laird, J.: A game semantics of the asynchronous π-calculus. In: 16th International Conference on CONCUR 2005 - Concurrency Theory, pp. 51–65 (2005)
25. Levy, P.B.: Morphisms between plays. In: Lecture Slides, GaLoP (2013)
26. Melliès, P.-A.: Asynchronous games 1: a group-theoretic formulation of uniformity (2003). (Unpublished manuscript)
27. Melliès, P.-A.: Asynchronous games 2: the true concurrency of innocence. Theor. Comput. Sci. **358**(2–3), 200–228 (2006)
28. Melliès, P.-A.: Game semantics in string diagrams. In: Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, pp. 481–490 (2012)
29. Melliès, P.-A., Mimram, S.: Asynchronous games: innocence without alternation. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 395–411. Springer, Heidelberg (2007). doi:10.1007/978-3-540-74407-8_27

30. Melliès, P.-A., Mimram, S.: From asynchronous games to concurrent games (2008). (Unpublished manuscript)
31. Murawski, A.S., Tzevelekos, N.: Nominal game semantics. Found. Trends Program. Lang. **2**(4), 191–269 (2016)
32. Nickau, H.: Hereditarily sequential functionals. In: Nerode, A., Matiyasevich, Y.V. (eds.) LFCS 1994. LNCS, vol. 813, pp. 253–264. Springer, Heidelberg (1994). doi:10.1007/3-540-58140-5_25
33. Power, J., Thielecke, H.: Closed Freyd-and kappa-categories. In: Automata, Languages and Programming. In: 26th International Colloquium, ICALP 1999, pp. 625–634 (1999)
34. Rideau, S., Winskel, G.: Concurrent strategies. In: Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, pp. 409–418 (2011)
35. Sangiorgi, D.: pi-calculus, internal mobility, and agent-passing calculi. Theor. Comput. Sci. **167**(1&2), 235–274 (1996)
36. Tsukada, T., Ong, C.-H.L.: Nondeterminism in game semantics via sheaves. In: 30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, pp. 220–231 (2015)
37. Tsukada, T., Ong, C.-H.L.: Plays as resource terms via non-idempotent intersection types. In: Grohe, M., Koskinen, E., Shankar, N. (eds.) Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2016, New York, USA, July 5–8, 2016, pp. 237–246. ACM (2016)
38. Varacca, D., Yoshida, N.: Typed event structures and the linear pi-calculus. Theor. Comput. Sci. **411**(19), 1949–1973 (2010)

Local Model Checking in a Logic for True Concurrency

Paolo Baldan and Tommaso Padoan(✉)

Dipartimento di Matematica, Università di Padova, Padova, Italy
{baldan,padoan}@math.unipd.it

Abstract. We provide a model-checking technique for a logic for true concurrency, whose formulae predicate about events in computations and their causal dependencies. The logic, that represents the logical counterpart of history-preserving bisimilarity, is naturally interpreted over event structures. It includes minimal and maximal fixpoint operators and thus it can express properties of infinite computations. Global algorithms are not convenient in this setting, since the event structure associated with a system is typically infinite (even if the system is finite state), a fact that makes also the decidability of model-checking non-trivial. Focusing on the alternation free fragment of the logic, along the lines of some classical work for the modal mu-calculus, we propose a local model-checking algorithm. The algorithm is given in the form of a tableau system, for which, over a class of event structures satisfying a suitable regularity condition, we prove termination, soundness and completeness.

1 Introduction

When dealing with concurrent and distributed systems, a partial order approach to the semantics can be appropriate for providing a precise account of the behavioural steps and of their dependencies, like causality and concurrency. This is normally referred to as a true concurrent approach to the semantics and opposed to the so-called interleaving approach where concurrency of actions is reduced to the non-deterministic choice among their possible sequentializations. True concurrent models can be convenient also because, thanks to an explicit representation of concurrency, they provide some relief to the so-called state-space explosion problem in the analysis of concurrent systems, which instead occurs more severely in interleaving approaches (see, e.g., [1]).

A number of true concurrent behavioural equivalences have been defined which take into account different concurrency features of computations (see, e.g., [2]). On the logical side, various behavioural logics have been proposed capable of expressing causal properties of computations (see, e.g., [3–8] just to mention a few and [9,10] for more references) and corresponding model-checking problems have been considered (see, e.g., [11–15]).

Recently, the logical characterisation of true concurrent behavioural equivalences has received a renewed interest and corresponding event-based logics

Partially supported by the University of Padova project ANCORE.

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 407–423, 2017.
DOI: 10.1007/978-3-662-54458-7_24

have been introduced [9,10], interpreted over event structures [16]. Logic formulae include variables which can be bound to events in computations and describe their dependencies, namely causality and concurrency. The expressiveness of such logics is sufficient to provide a logical characterisation of the main behavioural equivalences in the true concurrent spectrum [2]. Hereditary history preserving (hhp-)bisimilarity [4], the finest equivalence in the spectrum in [2], corresponds to the full logics, i.e., two systems are hhp-bisimilar if and only if they satisfy the same logical formulae, and fragments can be identified corresponding to coarser behavioural equivalences. The corresponding model-checking problem, instead, has not been investigated. Decidability itself is an issue, since event structure models are infinite even for finite state systems and problems in this framework have often revealed to sit on the border between decidability and undecidability.

In this paper we focus on a fragment of the event-based logic in [10], referred to as $\mathcal{L}_{hp}$, corresponding to a classical equivalence in the spectrum, namely history preserving (hp-)bisimilarity [17–19]. The logic is extended with minimal and maximal fixpoint operators, in mu-calculus style, in order to express interesting properties of infinite computations. Hp-bisimilarity is known to be decidable for finite safe Petri nets [20,21]. However, the question remains open on whether the corresponding model checking problem for $\mathcal{L}_{hp}$ is decidable over some interesting class of systems.

We answer positively to the question, defining a model checking procedure for $\mathcal{L}_{hp}$ over a class of event structures satisfying a suitable regularity condition.

Since the model checking procedure acts on event structures which are normally infinite even when generated from finite state systems, global algorithms fully exploring the structure are not a convenient choice. We are naturally led to focus on local algorithms, in the style of [22], exploring only the part of a model needed to asses the property. Along the lines of [22], the model checking procedure is given in the form of a tableau system. In order to check whether a system model satisfies a given formula a set of proof trees is constructed by applying a suitable set of rules that reduce the satisfaction of a formula in a given state to the satisfaction of suitably generated subformulae. In this way, the state space is explored "on demand" only in the part relevant for deciding the satisfaction of the formula. The presence of fixpoint operators makes the issue of sound termination quite delicate and non-trivial already in the original approach that works on finite transition systems. In the setting of this paper, this is further complicated by the infiniteness of the event structure model of any non-trivial system.

In order to single out a setting that ensures termination, soundness and completeness of the model checking procedure we take two key choices. The first is the restriction to a class of event structures having a finitary flavour, which we call strongly regular event structures. Recall that regular event structures [23] are characterised by the fact that the number of sub-structures arising as residuals of the original event structure after some steps of computations is finite up to isomorphism. The intuition is that, after going sufficiently in depth, the event structure starts repeating cyclically. For strongly regular event structures the

requirement is strengthened by asking the finiteness of the residuals extended with an event from the past. This is important in our setting since $\mathcal{L}_{hp}$ formulae can express history-based properties that depend not only on the future but also on past events. The second choice is the use of fixpoints in an alternation free fashion: minimal and maximal fixpoints can be nested, but without creating mutual dependencies (technically, the propositional variable bound by a minimal fixpoint operator cannot be in the scope of a maximal fixpoint operator and vice versa). When dealing with finite structures, alternation freeness allows for efficient verification procedures [24]. Here the essential fact is that it ensures that, over finitely branching transition systems, the formulae are continuous in the sense of [25,26], hence fixpoints are reached in at most ω steps (higher ordinals are never needed).

The paper is organised as follows. In Sect. 2 we recall some basics on (prime) event structures and we define the regularity property of interest. In Sect. 3 we introduce the true concurrent logic $\mathcal{L}_{hp}$, its fixpoint extension and the alternation free fragment. In Sect. 4 we define our model checking procedure as a tableau system, and we prove its soundness, completeness and termination. Finally, in Sect. 5 we discuss some related work and outline directions of future research.

2 Event Structures and Regularity

Prime event structures [16] are a widely known model of concurrency. They describe the behaviour of a system in terms of events and dependency relations between such events. Throughout the paper $\mathbb{E}$ is a fixed countable set of events, Λ a finite set of labels ranged over by $\mathsf{a}, \mathsf{b}, \mathsf{c} \ldots$ and $\lambda : \mathbb{E} \to \Lambda$ a labelling function.

Definition 1 (Prime event structure). *A (Λ-labelled)* Prime event structure *(*PES*) is a tuple $\mathcal{E} = \langle E, \leq, \# \rangle$, where $E \subseteq \mathbb{E}$ is the set of* events *and $\leq$, $\#$ are binary relations on E, called* causality *and* conflict *respectively, such that:*

1. *$\leq$ is a partial order and $\lceil e \rceil = \{e' \in E \mid e' \leq e\}$ is finite for all $e \in E$;*
2. *$\#$ is irreflexive, symmetric and hereditary with respect to $\leq$, i.e., for all $e, e', e'' \in E$, if $e \# e' \leq e''$ then $e \# e''$.*

The PES*s $\mathcal{E}_1 = \langle E_1, \leq_1, \#_1 \rangle$, $\mathcal{E}_2 = \langle E_2, \leq_2, \#_2 \rangle$ are* isomorphic*, written $\mathcal{E}_1 \sim \mathcal{E}_2$, when there is a bijection $\iota : E_1 \to E_2$ such that for all $e_1, e_1' \in E_1$ it holds $e_1 \leq_1 e_1'$ iff $\iota(e_1) \leq_2 \iota(e_1')$ and $e_1 \#_1 e_1'$ iff $\iota(e_1) \#_2 \iota(e_1')$ and $\lambda(e_1) = \lambda(\iota(e_1))$.*

In the following, we will assume that the components of an event structure $\mathcal{E}$ are named as in the definition above, possibly with subscripts.

Definition 2 (Consistency, concurrency). *Let $\mathcal{E}$ be a* PES*. We say that $e, e' \in E$ are* consistent*, written $e \frown e'$, if $\neg(e \# e')$. A subset $X \subseteq E$ is called* consistent *if $e \frown e'$ for all $e, e' \in X$. We say that e and e' are* concurrent*, written $e \,||\, e'$, if $e \frown e'$ and $\neg(e \leq e')$, $\neg(e' \leq e)$.*

Causality and concurrency will be sometimes used on set of events. Given $X \subseteq E$ and $e \in E$, by $X < e$ we mean that for all $e' \in X$, $e' < e$. Similarly $X \,||\, e$, resp. $X \frown e$, means that for all $e' \in X$, $e' \,||\, e$, resp. $e' \frown e$.

The concept of (concurrent) computation for event structures is captured by the notion of configuration.

Definition 3 (Configuration). *Let $\mathcal{E}$ be a* PES. *A* configuration *in $\mathcal{E}$ is a finite consistent subset of events $C \subseteq E$ closed w.r.t. causality (i.e., $\lceil e \rceil \subseteq C$ for all $e \in C$). The set of finite configurations of $\mathcal{E}$ is denoted by $\mathcal{C}(\mathcal{E})$.*

The empty set of events $\emptyset$ is always a configuration, which can be interpreted as the initial state of the computation. The evolution of a system can be represented by a transition system where configurations are states.

Definition 4 (Transition system). *Let $\mathcal{E}$ be a* PES *and let $C \in \mathcal{C}(\mathcal{E})$. Given $e \in E \backslash C$ such that $C \cup \{e\} \in \mathcal{C}(\mathcal{E})$, and $X, Y \subseteq C$ with $X < e$, $Y \parallel e$ we write $C \xrightarrow{X, \overline{Y} < e}_{\lambda(e)} C \cup \{e\}$, possibly omitting X, Y or the label $\lambda(e)$. The set of* enabled *events at a configuration C is defined as $en(C) = \{e \in E \mid C \xrightarrow{e} C'\}$.*

Transitions are labelled by the executed event e. In addition, they can report its label $\lambda(e)$, a subset of causes X and a set of events $Y \subseteq C$ concurrent with e.

We already mentioned that the PES associated with a non-trivial system exhibiting a cyclic behaviour is infinite. We next introduce a class of PESs enjoying a finitary property referred to as strong regularity.

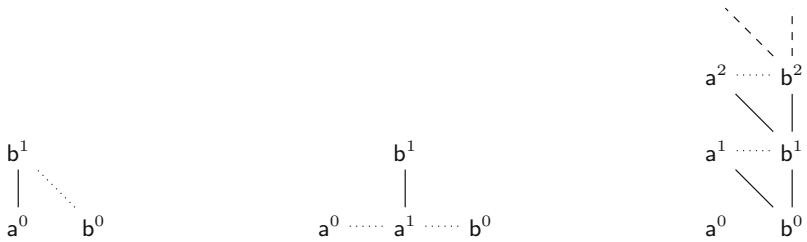
Definition 5 (Residual). *Let $\mathcal{E}$ be a* PES. *For a configuration $C \in \mathcal{C}(\mathcal{E})$, the* residual *of $\mathcal{E}$ after C, is defined as $\mathcal{E}[C] = \{e \in E \backslash C \;\mid\; C \frown e\}$.*

The residual of $\mathcal{E}$ can be seen as a PES, endowed with the restriction of the causality and conflict relations of $\mathcal{E}$. Intuitively, it represents the PES that remains to be executed after the computation expressed by C.

Recall that a PES $\mathcal{E}$ is *regular* [23] when the set of residuals $\{\mathcal{E}[C] \mid C \in \mathcal{C}(\mathcal{E})\}$ is finite up to isomorphism and there exists an integer k such that $|en(C)| \leq k$ for all $C \in \mathcal{C}(\mathcal{E})$. The first condition roughly means that there is a finite number of residuals over which the computation cycles. The second condition requires that the transition system of configurations is boundedly branching, with a finite bound. Here we strengthen the first condition by requiring the finiteness of the residuals extended with an event from the past. Given $C \in \mathcal{C}(\mathcal{E})$ and $e \in C$, we denote by $\mathcal{E}[C] \cup \{e\}$ the PES obtained from $\mathcal{E}[C]$ by adding event e with the causal dependencies it had in the original PES $\mathcal{E}$.

Definition 6 (Strong regularity). *A* PES *$\mathcal{E}$ is called* strongly regular *when the set $\{\mathcal{E}[C] \cup \{e\} \mid C \in \mathcal{C}(\mathcal{E}) \;\wedge\; e \in C\}$ is finite up to isomorphism of* PES*s and there exists an integer k such that $|en(C)| \leq k$ for all $C \in \mathcal{C}(\mathcal{E})$.*

Clearly, each strongly regular PES is regular. Strongly regular PESs can be shown, e.g., to include regular trace PESs, the class of event structures associated with finite safe Petri nets [23].



(a) $\mathcal{E}_1$ (b) $\mathcal{E}_2$ (c) $\mathcal{E}_3$

Fig. 1. Some examples of PESs.

Some simple PESs are depicted in Fig. 1. Graphically, dotted lines represent immediate conflicts and the causal partial order proceeds upwards along the straight lines. Events are denoted by their labels, possibly with superscripts. For instance, in the PES $\mathcal{E}_1$, there are two b-labelled events, b^0 and b^1 in conflict. The first is caused by the only a-labelled event a^0 and the other is concurrent with it. The infinite PES $\mathcal{E}_3$ corresponds to the CCS process $\mathsf{a} \parallel \mathsf{b}.C$ where $C = \mathsf{a} + \mathsf{b}.C$. It is strongly regular since it has seven (equivalence classes of) residuals extended with an event from the past $\mathcal{E}_3[\{\mathsf{a}^0\}] \cup \{\mathsf{a}^0\} = \mathcal{E}_3[\{\mathsf{b}^0\}] \cup \{\mathsf{b}^0\} = \mathcal{E}_3$, $\mathcal{E}_3[\{\mathsf{a}^0, \mathsf{b}^0\}] \cup \{\mathsf{a}^0\}$, $\mathcal{E}_3[\{\mathsf{a}^0, \mathsf{b}^0\}] \cup \{\mathsf{b}^0\}$, $\mathcal{E}_3[\{\mathsf{b}^0, \mathsf{a}^1\}] \cup \{\mathsf{b}^0\}$, $\mathcal{E}_3[\{\mathsf{b}^0, \mathsf{a}^1\}] \cup \{\mathsf{a}^1\}$, $\mathcal{E}_3[\{\mathsf{a}^0, \mathsf{b}^0, \mathsf{a}^1\}] \cup \{\mathsf{a}^0\} \simeq \mathcal{E}_3[\{\mathsf{a}^0, \mathsf{b}^0, \mathsf{a}^1\}] \cup \{\mathsf{a}^1\}$, and $\mathcal{E}_3[\{\mathsf{a}^0, \mathsf{b}^0, \mathsf{a}^1\}] \cup \{\mathsf{b}^0\}$.

3 A Logic for True Concurrency

In this section we introduce the syntax and the semantics of the logic for concurrency of interest in the paper. Originally defined in [10], the logic has formulae that predicate over executability of events in computations and on their dependency relations (causality and concurrency).

3.1 Syntax

As already mentioned, logic formulae include event variables from a fixed denumerable set *Var*, denoted by $x, y, \ldots$. In order to keep the notation simple, tuples of variables like $x_1, \ldots, x_n$ will be denoted by a corresponding boldface letter $\mathbf{x}$ and, abusing the notation, tuples will be often used as sets. The logic, in positive form, besides the standard propositional connectives $\wedge$ and $\vee$, includes diamond and box modalities. The formula $\langle\!|\mathbf{x}, \overline{\mathbf{y}} < \mathsf{a}\, z|\!\rangle\, \varphi$ holds when in the current configuration an a-labelled event e is enabled which causally depends on the events bound to the variables in $\mathbf{x}$ and is concurrent with those in $\mathbf{y}$. Event e is executed and bound to variable z, and then the formula φ must hold in the resulting configuration. Dually, $[\![\mathbf{x}, \overline{\mathbf{y}} < \mathsf{a}\, z]\!]\, \varphi$ is satisfied when all a-labelled events causally dependent on $\mathbf{x}$ and concurrent with $\mathbf{y}$ bring to a configuration where φ holds.

Fixpoint operators resort to propositional variables. In order to let them interact correctly with event variables, whose values can be passed from an iteration to the next one in the recursion, we use abstract propositions.

We fix a denumerable set $\mathcal{X}^a$ of *abstract propositions*, ranged over by X, Y, ..., that are intended to represent formulae possibly containing (unnamed) free event variables. Each abstract proposition has an arity $ar(X)$, which indicates the number of free event variables in X. An abstract proposition X can be turn into a formula by specifying a name for its free variables. For $\mathbf{x}$ such that $|\mathbf{x}| = ar(X)$, we write $X(\mathbf{x})$ to indicate the abstract proposition X whose free event variables are named $\mathbf{x}$. When $ar(X) = 0$ we will write X instead of $X(\epsilon)$ omitting the trailing empty tuple of variables. We call $X(\mathbf{x})$ a *proposition* and denote by $\mathcal{X}$ the set of all propositions.

Definition 7 (Syntax). *The syntax of $\mathcal{L}_{hp}$ over the sets of event variables Var, abstract propositions $\mathcal{X}^a$ and labels Λ is defined as follows:*

$$\begin{array}{rl} \varphi ::= X(\mathbf{x}) \mid \mathsf{T} \mid \varphi \wedge \varphi \mid \langle\!\langle \mathbf{x}, \overline{\mathbf{y}} < \mathsf{a}\, z \rangle\!\rangle \varphi \mid \nu X(\mathbf{x}).\varphi \\ \quad \mid \mathsf{F} \mid \varphi \vee \varphi \mid [\![\mathbf{x}, \overline{\mathbf{y}} < \mathsf{a}\, z]\!] \varphi \mid \mu X(\mathbf{x}).\varphi \end{array}$$

The free event variables of a formula φ are denoted by $fv(\varphi)$ and defined in the obvious way. Just note that the modalities act as binders for the variable representing the event executed, hence $fv(\langle\!\langle \mathbf{x}, \overline{\mathbf{y}} < \mathsf{a}\, z \rangle\!\rangle \varphi) = fv([\![\mathbf{x}, \overline{\mathbf{y}} < \mathsf{a}\, z]\!] \varphi) = (fv(\varphi) \backslash \{z\}) \cup \mathbf{x} \cup \mathbf{y}$. For formulae $\nu X(\mathbf{x}).\varphi$ and $\mu X(\mathbf{x}).\varphi$ we require that $fv(\varphi) = \mathbf{x}$. The free propositions in φ, i.e., the propositions not bound by μ or ν, are denoted by $fp(\varphi)$. When both $fv(\varphi)$ and $fp(\varphi)$ are empty we say that φ is *closed*.

The model checking procedure presented in the paper is shown to work in the so-called alternation free fragment of the logic. The idea is that propositions introduced in least (greatest) fixpoints should not occur free in nested greatest (least) fixpoints. More precisely, call an occurrence of an abstract proposition X a μ-proposition or ν-proposition when it is bound by a μ or ν operator, respectively. Then the definition is as follows.

Definition 8 (Alternation free formula). *A formula of $\mathcal{L}_{hp}$ is called* alternation free *when no subformula includes both a free μ-proposition and a free ν-proposition.*

An example of formula with alternation is the following

$$[\![\mathsf{a}\, x]\!] \mu Y(x).(\nu Z(x).[\![x < \mathsf{a}\, y]\!] Y(y) \wedge [\![x < \mathsf{b}\, y]\!] Z(y))$$

In fact, Z is a ν-proposition, Y is a μ-proposition and they both occur free in the subformula $[\![x < \mathsf{a}\, y]\!] Y(y) \wedge [\![x < \mathsf{b}\, y]\!] Z(y)$. It expresses that along every run starting with an a and consisting of an infinite causal chain of events, labelled a or b, only a finite number of a-labelled events can occur. Already for the propositional mu-calculus, it can be proved that alternation increases the expressiveness of the logic and also its complexity (see, e.g., [27]). Still, as argued in the same paper, the alternation free fragment is quite useful, in practice, as many behavioural properties of interest can be expressed in such fragment.

3.2 Semantics

Since the logic $\mathcal{L}_{hp}$ is interpreted over PESs, the satisfaction of a formula is defined with respect to a configuration C, representing the state of the computation, and a (total) function $\eta : \mathit{Var} \to E$, called an *environment*, that binds free variables in φ to events in C. Namely, if $\mathit{Env}_{\mathcal{E}}$ denotes the set of environments, the semantics of a formula will be a set of pairs in $\mathcal{C}(\mathcal{E}) \times \mathit{Env}_{\mathcal{E}}$. The semantics of $\mathcal{L}_{hp}$ also depends on a proposition environment providing a semantic interpretation for propositions.

Definition 9 (Proposition environments). *Let $\mathcal{E}$ be a* PES. *A* proposition environment *is a function $\pi : \mathcal{X} \to 2^{\mathcal{C}(\mathcal{E}) \times \mathit{Env}_{\mathcal{E}}}$ such that if $(C, \eta) \in \pi(X(\mathbf{x}))$ and $\eta'(\mathbf{y}) = \eta(\mathbf{x})$ pointwise, then $(C, \eta') \in \pi(X(\mathbf{y}))$. We denote by $\mathit{PEnv}_{\mathcal{E}}$ the set of proposition environments, ranged over by π.*

The condition imposed on proposition environments ensures that the semantics of a formula only depends on the events that the environment associates to its free variables and that it does not depend on the naming of the variables.

We can now give the semantics of the logic $\mathcal{L}_{hp}$. Given an event environment η and an event e we write $\eta[x \mapsto e]$ to indicate the updated environment which maps x to e. Similarly, for a proposition environment π and $S \subseteq \mathcal{C}(\mathcal{E}) \times \mathit{Env}_{\mathcal{E}}$, we write $\pi[Z(\mathbf{x}) \mapsto S]$ for the corresponding update.

Definition 10 (Semantics). *Let $\mathcal{E}$ be a* PES. *The denotation of a formula φ in $\mathcal{L}_{hp}$ is given by the function $\{\!|\cdot|\!\}^{\mathcal{E}} : \mathcal{L}_{hp} \to \mathit{PEnv}_{\mathcal{E}} \to 2^{\mathcal{C}(\mathcal{E}) \times \mathit{Env}_{\mathcal{E}}}$ defined inductively as follows, where we write $\{\!|\varphi|\!\}^{\mathcal{E}}_{\pi}$ instead of $\{\!|\varphi|\!\}^{\mathcal{E}}(\pi)$:*

$$\{\!|\mathsf{T}|\!\}^{\mathcal{E}}_{\pi} = \mathcal{C}(\mathcal{E}) \times \mathit{Env}_{\mathcal{E}} \qquad \{\!|\mathsf{F}|\!\}^{\mathcal{E}}_{\pi} = \emptyset \qquad \{\!|Z(\mathbf{y})|\!\}^{\mathcal{E}}_{\pi} = \pi(Z(\mathbf{y}))$$

$$\{\!|\varphi_1 \wedge \varphi_2|\!\}^{\mathcal{E}}_{\pi} = \{\!|\varphi_1|\!\}^{\mathcal{E}}_{\pi} \cap \{\!|\varphi_2|\!\}^{\mathcal{E}}_{\pi} \qquad \{\!|\varphi_1 \vee \varphi_2|\!\}^{\mathcal{E}}_{\pi} = \{\!|\varphi_1|\!\}^{\mathcal{E}}_{\pi} \cup \{\!|\varphi_2|\!\}^{\mathcal{E}}_{\pi}$$

$$\{\!|\langle\!\langle \mathbf{x}, \overline{\mathbf{y}} < \mathsf{a}\, z \rangle\!\rangle \varphi|\!\}^{\mathcal{E}}_{\pi} = \{(C, \eta) \mid \exists e.\, C \xrightarrow{\eta(\mathbf{x}), \overline{\eta(\mathbf{y})} < e}_{\mathsf{a}} C' \ \wedge\ (C', \eta[z \mapsto e]) \in \{\!|\varphi|\!\}^{\mathcal{E}}_{\pi}\}$$

$$\{\!|[\![\mathbf{x}, \overline{\mathbf{y}} < \mathsf{a}\, z]\!] \varphi|\!\}^{\mathcal{E}}_{\pi} = \{(C, \eta) \mid \forall e.\, C \xrightarrow{\eta(\mathbf{x}), \overline{\eta(\mathbf{y})} < e}_{\mathsf{a}} C' \ \Rightarrow\ (C', \eta[z \mapsto e]) \in \{\!|\varphi|\!\}^{\mathcal{E}}_{\pi}\}$$

$$\{\!|\nu Z(\mathbf{x}).\varphi|\!\}^{\mathcal{E}}_{\pi} = \mathit{gfp}(f_{\varphi, Z(\mathbf{x}), \pi}) \qquad \{\!|\mu Z(\mathbf{x}).\varphi|\!\}^{\mathcal{E}}_{\pi} = \mathit{lfp}(f_{\varphi, Z(\mathbf{x}), \pi})$$

where $f_{\varphi, Z(\mathbf{x}), \pi} : 2^{\mathcal{C}(\mathcal{E}) \times \mathit{Env}_{\mathcal{E}}} \to 2^{\mathcal{C}(\mathcal{E}) \times \mathit{Env}_{\mathcal{E}}}$ is the semantic function *of φ, $Z(\mathbf{x})$, π defined by $f_{\varphi, Z(\mathbf{x}), \pi}(S) = \{\!|\varphi|\!\}^{\mathcal{E}}_{\pi[Z(\mathbf{x}) \mapsto S]}$ and $\mathit{gfp}(f_{\varphi, Z(\mathbf{x}), \pi})$ (resp. $\mathit{lfp}(f_{\varphi, Z(\mathbf{x}), \pi})$) denotes the corresponding greatest (resp. least) fixpoint. When $(C, \eta) \in \{\!|\varphi|\!\}^{\mathcal{E}}_{\pi}$ we say that the* PES *$\mathcal{E}$ satisfies the formula φ in the configuration C and environments η, π, and we write $C, \eta \models^{\mathcal{E}}_{\pi} \varphi$.*

The semantics of boolean operators is as usual. The formula $\langle\!\langle \mathbf{x}, \overline{\mathbf{y}} < \mathsf{a}\, z \rangle\!\rangle \varphi$ holds in (C, η) when from configuration C there is an enabled a-labelled event e that is causally dependent on (at least) the events bound to the variables in $\mathbf{x}$ and concurrent with (at least) those bound to the variables in $\mathbf{y}$ and can be executed producing a new configuration $C' = C \cup \{e\}$ which, paired with the environment $\eta' = \eta[z \mapsto e]$, satisfies the formula φ. Dually, $[\![\mathbf{x}, \overline{\mathbf{y}} < \mathsf{a}\, z]\!] \varphi$ holds

when all a-labelled events executable from C, caused by $\mathbf{x}$ and concurrent with $\mathbf{y}$ bring to a configuration where φ is satisfied.

The fixpoints corresponding to the formulae $\nu Z(\mathbf{x}).\varphi$ and $\mu Z(\mathbf{x}).\varphi$ are guaranteed to exist by Knaster-Tarski theorem, since the set $2^{\mathcal{C}(\mathcal{E})\times Env_{\mathcal{E}}}$ ordered by subset inclusion is a complete lattice and the functions, of which the fixpoints are calculated, are monotonic.

For example, the formula $\langle\!|\mathsf{a}\, x|\!\rangle\langle\!|\overline{x} < \mathsf{b}\, y|\!\rangle\mathsf{T}$ says that we can execute an a-labelled event and a b-labelled event, concurrently. It is satisfied by all the PES in Fig. 1. The formula $\langle\!|\mathsf{a}\, x|\!\rangle(\langle\!|\overline{x} < \mathsf{b}\, y|\!\rangle\mathsf{T} \wedge \langle\!|x < \mathsf{b}\, z|\!\rangle\mathsf{T})$ expresses a simple "history dependent" property: it requires that, after the execution of an a-labelled event, one can choose between a concurrent and a causally dependent b-labelled event. This holds for $\mathcal{E}_1$ in Fig. 1a, while it is false on $\mathcal{E}_2$ in Fig. 1b where the choice is already determined by the execution of the a-labelled event.

As an example of property of infinite computations consider the formula $[\![\mathsf{b}\, x]\!]\nu Z(x).\langle\!|\mathsf{a}\, z|\!\rangle\langle\!|\overline{z} < \mathsf{a}\, y|\!\rangle\mathsf{T}\wedge[\![x < \mathsf{b}\, y]\!]Z(y)$, expressing that all non-empty causal chain of b-labelled events reach a state where is possible to execute two concurrent a-labelled events. This holds for the PES $\mathcal{E}_3$ in Fig. 1c. Now consider the formula $\mu X.\langle\!|\_\, z|\!\rangle X \vee \langle\!|\mathsf{b}\, x|\!\rangle\langle\!|x < \mathsf{a}\, y|\!\rangle\nu Y.\langle\!|\_\, z|\!\rangle Y$, where we use $\langle\!|\_\, z|\!\rangle\varphi$ as a shortcut for $\bigvee_{\mathsf{c}\in\Lambda}\langle\!|\mathsf{c}\, z|\!\rangle\,\varphi$. The formula, requiring the existence of an infinite run containing a b-labelled event immediately followed by a causally dependent a-labelled event, turns out to be false in the same PES. Intuitively this is because any a-labelled event causally dependent on a b-labelled event is in conflict with the rest of the infinite chain of events, and then, after its execution, the computation is guaranteed to terminate. A variant of the formula $\mu X.\langle\!|\_\, z|\!\rangle X \vee \langle\!|\mathsf{b}\, x|\!\rangle\langle\!|\overline{x} < \mathsf{a}\, y|\!\rangle\nu Y.\langle\!|\_\, z|\!\rangle Y$ requiring the existence of an infinite run containing a b-labelled event immediately followed by a concurrent a-labelled event, would be again true in $\mathcal{E}_3$. Observe that all formulae in the examples above are alternation free.

4 A Local Model Checker for the Logic

The model checker is given as a tableau system for testing whether an alternation free formula of the logic $\mathcal{L}_{hp}$ is satisfied by a semantic model of a system given in the form of a PES.

4.1 Constants and Definition Lists

As a first step, along the lines of [22], we extend the logic with propositional constants which are useful in defining the tableau rules.

Let $\mathcal{K}$ be a set of propositional constant symbols, ranged over by upper case letters like $U, V, W \ldots$. Similarly to abstract propositions, constants are meant to represent formulae of the logic, possibly containing (unnamed) free variables. Each constant U has an arity $ar(U)$, which indicates the number of free event variables in the associated formula. It can be used as a formula by specifying the names for its free variables, writing $U(\mathbf{x})$ where $|\mathbf{x}| = ar(U)$.

Definition 11 (Extended logic). *We denote by $\mathcal{L}^e_{hp}$ the logic defined as in Definition 7, with the addition of constants $U(\mathbf{x})$ as atomic formulae.*

Constants are interpreted at syntactical level, by means of a list of declarations that associates constants with formulae of the logic.

Definition 12 (Definition list). *A* definition list *is a sequence Δ of declarations $U_1(\mathbf{x}_1) = \varphi_1, \ldots, U_n(\mathbf{x}_n) = \varphi_n$, where $U_i \neq U_j$ whenever $i \neq j$ and each φ_k is a formula of the extended logic $\mathcal{L}^e_{hp}$ such that $|\mathbf{x}_k| = ar(U_k)$ and $fv(\varphi_k) = \mathbf{x}_k$, for $k \in \{1, \ldots, n\}$. We write $dom(\Delta) = \{U_1, \ldots, U_n\}$ for the set of constants declared in Δ, and $K(\varphi)$ for the set of constants appearing in φ. The definition list is* well-formed *if for all $k \in \{1, \ldots, n\}$ it holds that $K(\varphi_k) \subseteq \{U_1, \ldots, U_{k-1}\}$.*

Clearly, a prefix of a well-formed definition list is itself a well-formed definition list. Hereafter all definition lists will be implicitly assumed to be well-formed.

Definition 13 (Admissibility). *Let φ be a formula of the extended logic $\mathcal{L}^e_{hp}$. We say that a definition list Δ is* admissible *for φ if $K(\varphi) \subseteq dom(\Delta)$.*

A definition list is a sort of syntactical environment for constants, but, differently from environments, it is not total. The admissibility of Δ for φ simply means that each constant occurring in φ is declared in Δ, possibly with different names for its free variables. Given a constant $U \notin dom(\Delta)$ such that Δ is admissible for φ, $|fv(\varphi)| = ar(U)$ and $fv(\varphi) = \mathbf{x}$, we denote by $\Delta \cdot (U(\mathbf{x}) = \varphi)$ the definition list resulting by appending the declaration $U(\mathbf{x}) = \varphi$ to Δ.

Given a formula φ in the extended logic $\mathcal{L}^e_{hp}$ and an admissible definition list Δ, we can transform φ into a formula of the original logic $\mathcal{L}_{hp}$ by repeatedly replacing each constant with the corresponding definition. The substitution of a constant U according to its definition $U(\mathbf{x}) = \psi$ in a formula φ, denoted $\varphi[U := \psi]$, is the formula obtained from φ by replacing any occurrence of $U(\mathbf{y})$ with $\psi[\mathbf{y}/\mathbf{x}]$.

Definition 14 (Expansion). *Let φ be a formula in the extended logic $\mathcal{L}^e_{hp}$ and let Δ be $U_1(\mathbf{x}_1) = \psi_1, \ldots, U_n(\mathbf{x}_n) = \psi_n$, an admissible definition list for φ. The formula φ_Δ in $\mathcal{L}_{hp}$ is obtained by applying recursively n substitutions, starting from $\varphi_{(n)} = \varphi[U_n := \psi_n]$ and then $\varphi_{(i-1)} = \varphi_{(i)}[U_{i-1} := \psi_{i-1}]$, until $\varphi_{(1)}$ is calculated. Then $\varphi_\Delta = \varphi_{(1)}$.*

We will often write $\Delta(U(\mathbf{y}))$ to indicate the formula of the extended logic associated with $U(\mathbf{y})$ in Δ. Free variables and free propositions of a formula φ in the extended logic, in an admissible definition list Δ, are defined as before, with the addition of the clauses $fv(U(\mathbf{y})) = \mathbf{y}$ and $fp(U(\mathbf{y})) = fp(\Delta(U(\mathbf{y})))$.

Concerning the semantics, we say that a PES $\mathcal{E}$ satisfies the formula φ with the admissible definition list Δ in the configuration C and environments η, π, written $C, \eta, \Delta \models^{\mathcal{E}}_{\pi} \varphi$, when $(C, \eta) \in \{\!|\varphi_\Delta|\!\}^{\mathcal{E}}_{\pi}$.

4.2 Tableau Rules

The tableau system works on judgements $\Gamma \models^{\mathcal{E}} \varphi$, where $\Gamma = \langle C, \eta, \Delta \rangle$, referred to as a *context*, is a tuple consisting of a configuration $C \in \mathcal{C}(\mathcal{E})$, an environment η and a definition list Δ, admissible for φ, such that φ_Δ is closed. It consists in a set of rules of the form

$$\frac{C, \eta, \Delta \models^{\mathcal{E}} \varphi}{C_1, \eta_1, \Delta_1 \models^{\mathcal{E}} \varphi_1 \ \ldots \ C_k, \eta_k, \Delta_k \models^{\mathcal{E}} \varphi_k} \ \delta$$

where $k > 0$ and δ is a possible side condition required to hold. The intuition is that the validity of the premise sequent reduces to the validity of its consequents. The index $\mathcal{E}$, when the model $\mathcal{E}$ is clear from the context, will be dropped.

In [22,28], the finiteness of the model is an essential ingredient that concurs to the finiteness of the tableaux. In our case, as already mentioned, the PES model is commonly infinite, even for finite-state systems. However, working on strongly regular PESs we have that (1) only a finite part of the model is used in every step of the tableau construction, thanks to the fact that strongly regular PESs are boundedly branching and (2) after going sufficiently in depth, the PES starts "repeating" cyclically the same structure. These facts will allow to conclude that the satisfaction of a formula can be determined by checking only a finite part of the PES (as formally proved later in Sect. 4.4).

The tableau rules for the logic $\mathcal{L}^{e}_{hp}$, are reported in Table 1. The rules for the propositional connectives are straightforward. They reduce the satisfaction of the formula in the premise to the satisfaction of subformulae in an obvious way. For instance $\varphi \vee \psi$ is satisfied when either φ is satisfied or ψ is satisfied. The context is not altered.

Similarly, the rules for the modal operators generate sequents involving the subformulae after the modal operators with a context that changes according to the semantics. The formula $\langle\!\langle \mathbf{x}, \overline{\mathbf{y}} < \mathsf{a}\, z \rangle\!\rangle \varphi$ holds when there is at least a transition leading to a context where φ is satisfied, while $[\![\mathbf{x}, \overline{\mathbf{y}} < \mathsf{a}\, z]\!] \varphi$ is satisfied when all transitions lead to a context where φ is satisfied.

For each kind of fixpoint formula $\alpha Z(\mathbf{x}).\varphi$, with $\alpha \in \{\mu, \nu\}$, there are two rules. The first rule introduces a constant, which is added to the definition list and used as a formula in the consequent. The second rule unfolds the fixpoint by unrolling the associated constant and also reassigns its variables by updating the environment. The side condition involves an unspecified part γ, the so-called *stop* condition, that prevents the reduction to continue unboundedly and will be described in the next section.

The tableau rules are backwards sound, namely the premise is true if all the consequents are true, a property that will play a basic role in Sect. 4.4.

Lemma 1 (Backwards soundness). *Every rule of the tableau system is backwards sound.*

4.3 The Stop Condition

In order to ensure the finiteness of the tableaux generated by a formula, the unfolding of the fixpoints has to be stopped when a context is reached that is

Table 1. The tableau rules for logic $\mathcal{L}^{e}_{hp}$.

$$\frac{C,\eta,\Delta \models \varphi \wedge \psi}{C,\eta,\Delta \models \varphi \quad C,\eta,\Delta \models \psi}$$

$$\frac{C,\eta,\Delta \models \varphi \vee \psi}{C,\eta,\Delta \models \varphi} \qquad \frac{C,\eta,\Delta \models \varphi \vee \psi}{C,\eta,\Delta \models \psi}$$

$$\frac{C,\eta,\Delta \models \langle\!\langle \mathbf{x},\overline{\mathbf{y}} < \mathsf{a}\, z \rangle\!\rangle \varphi}{C',\eta[z \mapsto e],\Delta \models \varphi} \quad \exists\, e.\ C \xrightarrow{\eta(\mathbf{x}),\overline{\eta(\mathbf{y})} < e}_{\mathsf{a}} C'$$

$$\frac{C,\eta,\Delta \models [\![\mathbf{x},\overline{\mathbf{y}} < \mathsf{a}\, z]\!] \varphi}{C_1,\eta_1,\Delta \models \varphi \ \ldots\ C_n,\eta_n,\Delta \models \varphi}$$

where $\{(C_1,\eta_1),\ldots,(C_n,\eta_n)\} = \{(C',\eta[z \mapsto e]) \mid \exists\, e.\ C \xrightarrow{\eta(\mathbf{x}),\overline{\eta(\mathbf{y})} < e}_{\mathsf{a}} C'\}$

$$\frac{C,\eta,\Delta \models \alpha Z(\mathbf{x}).\varphi}{C,\eta,\Delta' \models U(\mathbf{x})} \quad \Delta' \text{ is } \Delta \cdot (U(\mathbf{x}) = \alpha Z(\mathbf{x}).\varphi) \text{ for } \alpha \in \{\nu,\mu\}$$

$$\frac{C,\eta,\Delta \models U(\mathbf{y})}{C,\eta',\Delta \models \varphi[Z \coloneqq U(\mathbf{y})]} \quad \neg\gamma \text{ and } \Delta(U(\mathbf{x})) = \alpha Z(\mathbf{x}).\varphi \text{ for } \alpha \in \{\nu,\mu\} \text{ and } \eta' = \eta[\mathbf{x} \mapsto \eta(\mathbf{y})],$$

equivalent, in a suitable sense to be defined, to a context occurring in an ancestor of the current node of the tableau, for the same fixpoint formula. The notion of equivalence should prevent the tableau generation to continue infinitely but it should be chosen carefully not to break the soundness of the technique.

Surely two contexts $\Gamma = \langle C,\eta,\Delta\rangle$ and $\Gamma' = \langle C',\eta',\Delta'\rangle$ for a formula φ, in order to be considered equivalent, must have isomorphic futures, i.e., the residuals $\mathcal{E}[C]$ and $\mathcal{E}[C']$ are isomorphic as PESs. This is not sufficient, though, since φ can express history dependent properties that relates the future with the past events. Therefore we additionally ask that event variables of φ are mapped to events in C and C', respectively, which have the same relations (causality and concurrency) with the corresponding futures.

In order to formalise this intuition we need to set up some notions.

Definition 15 (Contextualized residual). *Let $\mathcal{E}$ be a* PES*. Given a configuration $C \in \mathcal{C}(\mathcal{E})$, an environment η, a finite set of variables $\mathbf{x} \subseteq Var$, we refer to $\mathcal{E}[C,\eta,\mathbf{x}] = \langle \mathcal{E}[C],\eta,\mathbf{x}\rangle$ as the $(\eta,\ \mathbf{x})$-*contextualised residual *of $\mathcal{E}$ after C.*

The contextualised residuals $\mathcal{E}[C,\eta,\mathbf{x}]$ and $\mathcal{E}[C',\eta',\mathbf{x}']$ are isomorphic when $\mathbf{x} = \mathbf{x}'$ and there is an isomorphism between $\mathcal{E}[C]$ and $\mathcal{E}[C']$ "compatible" with the way environments map the variables in $\mathbf{x}$ into events.

Definition 16 (Isomorphism of contextualized residuals). *Let $\mathcal{E}$ be a* PES. *Two contextualised residuals $\mathcal{E}[C_1, \eta_1, \mathbf{x}_1]$ and $\mathcal{E}[C_2, \eta_2, \mathbf{x}_2]$ are isomorphic, written $\mathcal{E}[C_1, \eta_1, \varphi] \sim \mathcal{E}[C_2, \eta_2, \varphi]$, when $\mathbf{x}_1 = \mathbf{x}_2$ and there is an isomorphism $\iota : \mathcal{E}[C_1] \to \mathcal{E}[C_2]$ such that for any $x \in \mathbf{x}_1$, $e_1 \in \mathcal{E}[C_1]$ it holds $\eta_1(x) \leq_1 e_1 \iff \eta_2(x) \leq_2 \iota(e_1)$.*

A key observation is that isomorphic contextualised residuals satisfy exactly the same formulae, in the sense of the following theorem.

Lemma 2 (Equivalent contexts, logically). *Let $\mathcal{E}$ be a* PES, *let φ be a formula and let $\Gamma_1 = \langle C_1, \eta_1, \Delta_1 \rangle$ and $\Gamma_2 = \langle C_2, \eta_2, \Delta_2 \rangle$ be contexts, where $C_1, C_2 \in \mathcal{C}(\mathcal{E})$, $\Delta_1 \subseteq \Delta_2$ and Δ_1 (hence Δ_2) admissible for φ. For all proposition environments π_1, π_2 such that $\forall\ Z \in fp(\varphi_{\Delta_1})$, $(C_1, \eta_1) \in \pi_1(Z(\mathbf{y})) \iff (C_2, \eta_2) \in \pi_2(Z(\mathbf{y}))$, if $\mathcal{E}[C_1, \eta_1, fv(\varphi)] \sim \mathcal{E}[C_2, \eta_2, fv(\varphi)]$ then $\Gamma_1 \models^{\mathcal{E}}_{\pi_1} \varphi \iff \Gamma_2 \models^{\mathcal{E}}_{\pi_2} \varphi$.*

Note that the condition on the proposition environments π_1, π_2 is vacuously satisfied for formulae in the sequents of a tableau. In fact, the sequent labelling the root contains a closed formula and thus, by construction, formulae do not contain free propositions and neither do those associated to constants.

The results above motivate the definition of the stop condition.

Definition 17 (Stop condition). *The* stop condition γ *for a rule where the premise is $C, \eta, \Delta \models U(\mathbf{y})$, is as follows:*

$$\textit{There is an ancestor of the node labelled with } C', \eta', \Delta' \models U(\mathbf{z}), \textit{ such that } \mathcal{E}[C', \eta'[\mathbf{y} \mapsto \eta'(\mathbf{z})], \mathbf{y}] \sim \mathcal{E}[C, \eta, \mathbf{y}].$$

Informally, the stop condition holds when in a previous step of the construction of the tableau the same constant has been unfolded in a context equivalent to the current one, possibly after some renaming of variables. Hence we can safely avoid to continue along this path. Instead, when the stop condition fails, it makes sense to further unroll the fixpoint since the current context is still "different enough" from those previously encountered.

4.4 Model Checking a Formula Through Tableaux

For checking whether a closed formula φ in $\mathcal{L}_{hp}$ is satisfied by a PES $\mathcal{E}$, we proceed by building a tableau for the sequent $\emptyset, \eta, \emptyset \models^{\mathcal{E}} \varphi$, where η is any environment (irrelevant since the formula does not have free variables). A maximal tableau is a proof tree where all leaves are labelled by sequents to which no rule applies.

We next clarify when a maximal tableau is considered successful.

Definition 18 (Successful tableau). *A* successful *tableau is a finite maximal tableau where every leaf is labelled by a sequent $C, \eta, \Delta \models^{\mathcal{E}} \varphi$ such that:*

1. $\varphi = \mathsf{T}$; *or*
2. $\varphi = [\![\mathbf{x}, \overline{\mathbf{y}} < \mathsf{a}\, z]\!]\, \psi$; *or*
3. $\varphi = U(\mathbf{y})$ *and* $\Delta(U(\mathbf{x})) = \nu Z(\mathbf{x}).\psi$.

We will prove that in a successful tableau all leaves are labelled by true sequents, a fact that, together with backwards soundness of the rules (Lemma 1), will guarantee the truth of the sequent labelling the root.

Note that the choice of the rule to be applied at a step of the construction of a tableau is non-deterministic in the case of $\langle\!|\mathbf{x}, \overline{\mathbf{y}} < \mathsf{a}\, z|\!\rangle\, \varphi$ and $\varphi \vee \psi$. This means that there can be various maximal tableaux for the same sequent. However, when we work on strongly regular PESs, the fact that they are boundedly branching ensures that at each step the number of possible non-deterministic choices is finite and bounded. In turn this is used to deduce that there can be only a finite number of maximal tableaux for a sequent, up to renaming of constants.

We next focus the finiteness issue and then move on to the soundness and completeness of the technique.

Finiteness. We first aim at proving that all tableaux for a sequent $\emptyset, \eta, \emptyset \models^{\mathcal{E}} \varphi$ are finite. A first basic observation is that an infinite tableau for a sequent $C, \eta, \Delta \models^{\mathcal{E}} \varphi$ necessarily includes a path where the same constant is unfolded infinitely many times.

Lemma 3 (Infinite repetitions of constants in infinite tableaux). *Given an infinite tableau for a sequent $C, \eta, \Delta \models^{\mathcal{E}} \varphi$ there exists a constant U and an infinite path in the tableau such that U occurs in an unfolding rule infinitely many often, possibly each time with different event variables $\mathbf{x}_i$.*

Along the lines of [22], the proof relies on the introduction of a notion of degree for a sequent, that intuitively estimates the length of the longest path in the tableau that starts from the sequent itself and ends at the first sequent whose formula is a constant previously introduced (not new) or at a leaf. We already observed that the fact that strongly regular PESs are boundedly branching implies that also the constructed tableaux are finitely branching. Then, by König lemma, an infinite tableau necessarily include an infinite path. The observation that the degree is finite, non-negative and it decreases along a path until a previously introduced constant is met again, allows one to conclude.

A crucial observation is now that, for strongly regular PESs, the number of contextualised residuals is finite up to isomorphism. Using this fact, if there were an infinite path in a tableau and thus, by Lemma 3, a constant occurring infinitely often in such a path, then the constant would occur infinitely often within isomorphic contextualised residuals, leading to a contradiction. In fact, at the first repetition the stop condition (see Definition 17) would have prescribed of terminating the branch.

Theorem 1 (Finite number of contextualised residuals). *Let $\mathcal{E}$ be a strongly regular* PES *and let $\mathbf{x} \subseteq Var$ be a finite set of variables. Then the class of $(\eta, \mathbf{x})$-contextualised residuals of $\mathcal{E}$ after C, i.e., $\{\mathcal{E}[C, \eta, \mathbf{x}] \mid \eta \in Env_{\mathcal{E}} \wedge C \in \mathcal{C}(\mathcal{E})\}$ is finite up to isomorphism.*

We can finally deduce the finiteness of the tableaux for a sequent that in turn implies that the number of tableaux is finite (up to constant renaming). This fact is essential for termination of the model checking procedure.

Theorem 2 (Tableaux finiteness). *Given a strongly regular* PES $\mathcal{E}$ *and a formula* φ*, every tableau for a sequent* $\Gamma \models^{\mathcal{E}} \varphi$ *is finite. Hence the number of tableaux for* $\Gamma \models^{\mathcal{E}} \varphi$ *is finite up to constant renaming.*

Soundness and Completeness. We finally prove the soundness and completeness of the tableau system. For this we use the possibility of reducing the satisfaction of a formula to the satisfaction of a finite approximant. While on finite models this is immediate, here we need the (co)continuity of the semantic functions associated to formulae (see Definition 10) ensuring that the fixpoints will be reached in at most ω steps.

Let X be a set. A subset $A \subseteq 2^X$ is called *directed* if for any $S_1, S_2 \in A$ there exists $S \in A$ such that $S_1, S_2 \subseteq S$. A function $f : 2^X \to 2^X$ is *continuous* when for any directed set $A \subseteq 2^X$ it holds that $f(\bigcup A) = \bigcup\{f(S) \mid S \in A\}$. We call it *co-continuous* when it is continuous in the reverse (superset) order.

The semantic functions associated with formulae of the logic $\mathcal{L}_{hp}$ (see Definition 10) are neither continuous nor co-continuous in general. However, when requiring alternation freeness, μ-formulae, i.e., formulae where all ν-operators are in the scope of a μ-operator, are continuous and ν-formulae, defined dually, are co-continuous.

Lemma 4 ((Co-)continuous semantic operators). *Let* $\mathcal{E}$ *be a* PES*. Given an alternation free* μ*-formula* ψ*, a proposition* $Z(\mathbf{z}) \in \mathcal{X}$ *and a proposition environment* $\pi \in PEnv_{\mathcal{E}}$*, the semantic function* $f_{\psi, Z(\mathbf{z}), \pi}$ *is continuous. Dually, if* ψ *is an alternation free* ν*-formula the function* $f_{\psi, Z(\mathbf{z}), \pi}$ *is co-continuous.*

Note that the continuous fragment here is wider than that in [25] as, in particular, it includes the box modality. The difference is motivated by the fact that strongly regular PESs are finitely branching.

By Kleene Theorem the least fixpoint of a continuous function on a lattice requires up to ω iterations to be reached, namely if $f : 2^X \to 2^X$ is continuous then $lfp(f) = \bigcup_{i \in \mathbb{N}} f^i(\emptyset)$. Similarly, if f is co-continuous $gfp(f) = \bigcap_{i \in \mathbb{N}} f^i(X)$. This fact plays a role in the proof of the main result below.

Theorem 3 (Soundness and completeness of the tableaux system). *Given a strongly regular* PES $\mathcal{E}$ *and a closed alternation free formula* φ *of* $\mathcal{L}_{hp}$*, a sequent* $C, \eta, \emptyset \models^{\mathcal{E}} \varphi$ *has a successful tableau if and only if* $(C, \eta) \in \{\!|\varphi|\!\}^{\mathcal{E}}$*.*

5 Conclusions

We provided a tableau system for model checking the alternation free fragment of a logic for true concurrency $\mathcal{L}_{hp}$ over strongly regular PESs, proving finiteness of the tableaux and soundness and completeness of the rule system. Such results, together with Theorem 2, that ensures that a sequent has a(n essentially) finite number of tableaux, leads to a decision procedure for the alternation free fragment of $\mathcal{L}_{hp}$ over strongly regular event structures. A concrete procedure

requires the effectiveness of the transition relation over configurations and of the equivalence of contextualised residuals, that we have if we focus on regular trace PESs. Indeed, a natural instantiation of the model checking procedure can be given on finite safe Petri nets. For space reasons its presentation is delayed to the full version of the paper.

While regular trace event structures can be shown to be strongly regular, we still do not know whether also the converse holds. Some preliminary investigations lead us to conjecture that this is the case.

Soundness and completeness of the tableau system rely on the (co)continuity of the semantic functions associated with the formulae that we obtained by considering the alternation free fragment of the logic. While continuity fails outside this fragment, we conjecture that, exploiting the regularity property of the PES models, a sort of continuity up to isomorphism can be proved, allowing to extend the procedure to the full logic $\mathcal{L}_{hp}$.

Another open issue concerns the possibility of generalising the tableau-based technique to the full logic $\mathcal{L}$ in [10]. This is quite challenging: the full logic $\mathcal{L}$ induces a behavioural equivalence – hhp-bisimilarity – which is known to be undecidable already for finite state Petri nets [29]. Note that this does not imply undecidability of the corresponding model checking problem. On the semantic side one could try to relax the restriction to strongly regular PESs. However, we tend to believe that few results can be obtained when the realm of regular structures is abandoned.

Model checking on event structures has been considered by several other authors. In [30] a finite representation of the PES corresponding to the behaviour of a suitable class of programs is proposed, showing how discrete event logics can be model-checked on such structures. The paper [15] shows that first order logic and a restricted form of monadic second order (MSO) logic are decidable on regular trace event structures. The fact that the mu-calculus, in the propositional case, corresponds to the bisimulation invariant fragment of MSO logic [31] suggests the possibility of exploiting the mentioned result in our setting. This would require an encoding of $\mathcal{L}_{hp}$ into the MSO logic of [15]. More generally, understanding which are the bisimulation invariant fragments of MSO over event structures, with respect to the various concurrent bisimulations, represents an interesting program in itself. The work in [14] develops higher-order games as a mean for local model-checking a concurrent logic over partial order semantics. Despite the fact that such logic is different (and of incomparable expressive power with ours as explained in [10]), exploring the possibility of adopting a game-theoretical approach in our setting appears an interesting venue of further research.

Finally, a lot of work exists in the literature for the propositional mu-calculus, proposing efficient automata-based model checking techniques [32,33], that reduce the model checking problem to the non-emptiness problem of parity tree automata. Trying to develop a similar approach for the logic $\mathcal{L}_{hp}$, with the idea that the automata would somehow inherit the regularities in the structure of the PESs, represents a stimulating direction of future research.

References

1. Esparza, J., Heljanko, K.: Unfoldings - A Partial order Approach to Model Checking. EACTS Monographs. Springer, Heidelberg (2008)
2. van Glabbeek, R., Goltz, U.: Refinement of actions and equivalence notions for concurrent systems. Acta Inform. **37**(4/5), 229–327 (2001)
3. Nicola, R., Ferrari, G.L.: Observational logics and concurrency models. In: Nori, K.V., Veni Madhavan, C.E. (eds.) FSTTCS 1990. LNCS, vol. 472, pp. 301–315. Springer, Heidelberg (1990). doi:10.1007/3-540-53487-3_53
4. Bednarczyk, M.A.: Hereditary history preserving bisimulations or what is the power of the future perfect in program logics. Technical report, Polish Academy of Sciences (1991)
5. Pinchinat, S., Laroussinie, F., Schnoebelen, P.: Logical characterization of truly concurrent bisimulation. Technical report 114, LIFIA-IMAG, Grenoble (1994)
6. Penczek, W.: Branching Time and Partial Order in Temporal Logics. Time and Logic: A Computational Approach, pp. 179–228. UCL Press, London (1995)
7. Nielsen, M., Clausen, C.: Games and logics for a noninterleaving bisimulation. Nordic J. Comput. **2**(2), 221–249 (1995)
8. Bradfield, J., Fröschle, S.: Independence-friendly modal logic and true concurrency. Nordic J. Comput. **9**(1), 102–117 (2002)
9. Phillips, I., Ulidowski, I.: Event identifier logic. Math. Struct. Comput. Sci. **24**(2), 1–51 (2014)
10. Baldan, P., Crafa, S.: A logic for true concurrency. J. ACM **61**(4), 24:1–24:36 (2014)
11. Alur, R., Peled, D., Penczek, W.: Model-checking of causality properties. In: Proceedings of LICS 1995, pp. 90–100. IEEE Computer Society (1995)
12. Gutierrez, J., Bradfield, J.: Model-checking games for fixpoint logics with partial order models. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 354–368. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04081-8_24
13. Gutierrez, J.: Logics and bisimulation games for concurrency, causality and conflict. In: Alfaro, L. (ed.) FoSSaCS 2009. LNCS, vol. 5504, pp. 48–62. Springer, Heidelberg (2009). doi:10.1007/978-3-642-00596-1_5
14. Gutierrez, J.: On bisimulation and model-checking for concurrent systems with partial order semantics. Ph.D. thesis, University of Edinburgh (2011)
15. Madhusudan, P.: Model-checking trace event structures. In: Proceedings of LICS 2003, pp. 371–380. IEEE Computer Society (2003)
16. Winskel, G.: Event structures. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) ACPN 1986. LNCS, vol. 255, pp. 325–392. Springer, Heidelberg (1987). doi:10.1007/3-540-17906-2_31
17. Best, E., Devillers, R., Kiehn, A., Pomello, L.: Fully concurrent bisimulation. Acta Inform. **28**, 231–261 (1991)
18. Rabinovich, A.M., Trakhtenbrot, B.A.: Behaviour structures and nets. Fundam. Inform. **11**, 357–404 (1988)
19. Degano, P., Nicola, R., Montanari, U.: Partial orderings descriptions and observations of nondeterministic concurrent processes. In: Bakker, J.W., Roever, W.-P., Rozenberg, G. (eds.) REX 1988. LNCS, vol. 354, pp. 438–466. Springer, Heidelberg (1989). doi:10.1007/BFb0013030
20. Vogler, W.: Deciding history preserving bisimilarity. In: Albert, J.L., Monien, B., Artalejo, M.R. (eds.) ICALP 1991. LNCS, vol. 510, pp. 495–505. Springer, Heidelberg (1991). doi:10.1007/3-540-54233-7_158

21. Montanari, U., Pistore, M.: History-dependent automata: an introduction. In: Bernardo, M., Bogliolo, A. (eds.) SFM-Moby 2005. LNCS, vol. 3465, pp. 1–28. Springer, Heidelberg (2005). doi:10.1007/11419822_1
22. Stirling, C., Walker, D.: Local model checking in the modal mu-calculus. Theor. Comput. Sci. **89**(1), 161–177 (1991)
23. Thiagarajan, P.S.: Regular event structures and finite Petri Nets: a conjecture. In: Brauer, W., Ehrig, H., Karhumäki, J., Salomaa, A. (eds.) Formal and Natural Computing - Essays Dedicated to Grzegorz Rozenberg. LNCS, vol. 2300, pp. 244–253. Springer, Heidelberg (2002). doi:10.1007/3-540-45711-9_14
24. Cleaveland, R., Steffen, B.: A linear-time model-checking algorithm for the alternation-free modal mu-calculus. Form. Methods Syst. Des. **2**(2), 121–147 (1993)
25. Fontaine, G.: Continuous fragment of the mu-calculus. In: Kaminski, M., Martini, S. (eds.) CSL 2008. LNCS, vol. 5213, pp. 139–153. Springer, Heidelberg (2008). doi:10.1007/978-3-540-87531-4_12
26. Carreiro, F., Facchini, A., Venema, Y., Zanasi, F.: Weak MSO: automata and expressiveness modulo bisimilarity. In: Henzinger, T.A., Miller, D. (eds.) Proceedings of CSL-LICS 2014, pp. 27:1–27:27. ACM Press (2014)
27. Bradfield, J., Stirling, C.: Modal mu-calculi. In: Blackburn, P., van Benthem, J., Wolter, F. (eds.) Handbook of Modal Logic, pp. 721–756. Elsevier, New York (2006)
28. Clarke, E.M., Schlingloff, B.H.: Model checking. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning. Elsevier, Amsterdam (2001)
29. Jurdzinski, M., Nielsen, M., Srba, J.: Undecidability of domino games and hhp-bisimilarity. Inf. Comput. **184**(2), 343–368 (2003)
30. Penczek, W.: Model-checking for a subclass of event structures. In: Brinksma, E. (ed.) TACAS 1997. LNCS, vol. 1217, pp. 145–164. Springer, Heidelberg (1997). doi:10.1007/BFb0035386
31. Janin, D., Walukiewicz, I.: On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In: Montanari, U., Sassone, V. (eds.) CONCUR 1996. LNCS, vol. 1119, pp. 263–277. Springer, Heidelberg (1996). doi:10.1007/3-540-61604-7_60
32. Emerson, E.A., Jutla, C.S., Sistla, A.P.: On model checking for the μ-calculus and its fragments. Theor. Comput. Sci. **258**(1–2), 491–522 (2001)
33. Streett, R., Emerson, E.A.: An automata theoretic decision procedure for the propositional mu-calculus. Inf. Comput. **81**(3), 249–264 (1989)

The Paths to Choreography Extraction

Luís Cruz-Filipe(✉), Kim S. Larsen, and Fabrizio Montesi

University of Southern Denmark, Odense, Denmark
{lcf,kslarsen,fmontesi}@imada.sdu.dk

Abstract. Choreographies are global descriptions of interactions among concurrent components, most notably used in the settings of verification and synthesis of correct-by-construction software. They require a top-down approach: programmers first write choreographies, and then use them to verify or synthesize their programs. However, most software does not come with choreographies yet, which prevents their application. To attack this problem, previous work investigated choreography extraction, which automatically constructs a choreography that describes the behavior of a given set of programs or protocol specifications.

We propose a new extraction methodology that improves on the state of the art: we can deal with programs that are equipped with state and internal computation; time complexity is dramatically better; and we capture programs that work by exploiting asynchronous communication.

1 Introduction

Choreographies are global descriptions of interactions among components. They have been used as a basis for different models and tools that aim at tackling the complexity of modern software, where separate units – such as processes, objects, and threads – interact to reach a common goal [3,25].

Two lines of research are of particular interest. In *choreography specifications*, choreographies specify interaction protocols, e.g., multiparty session types [17]. In *choreographic programming* [20], choreographies are programs that define the behavior of concurrent algorithms [13] and/or distributed systems [5,6,14]. The key feature of these works is EndPoint Projection (EPP), a procedure that translates choreographies to correct endpoint behaviors in lower-level models. For choreography specifications, EPP generates the local specifications of each participant; these specifications can then be used for verification, to check whether some programs implement their role in the protocol correctly and will thus interact without problems at runtime [17]. In choreographic programming, instead, EPP generates correct-by-construction implementations in a model of executable code (program synthesis), typically given in terms of a process calculus [6].

Montesi was supported by CRC (Choreographies for Reliable and efficient Communication software), grant DFF–4005-00304 from the Danish Council for Independent Research. Cruz-Filipe and Larsen were supported in part by the Danish Council for Independent Research, Natural Sciences, grant DFF-1323-00247.

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 424–440, 2017.
DOI: 10.1007/978-3-662-54458-7_25

EPP implements a top-down development methodology: developers first write choreographies and then use the output mechanically generated by EPP. However, there are scenarios where this methodology is not applicable; for example:

– Analysis or integration of legacy software: either code developed previously, or new code written in a technology without support for choreographies.
– Updates: endpoint programs generated by EPP can later be updated locally (e.g., for configuration or optimizations). Since the original choreography is not automatically updated, rerunning EPP loses these changes.

To attack these issues, previous work investigated a procedure to infer choreographies from arbitrary endpoint descriptions. We call this procedure *choreography extraction*. To the best of our knowledge, the current reference for extracting choreography specifications is [19], where graphical choreographies that represent protocol specifications are extracted from communicating automata [4]. Instead, the state of the art for extraction in choreographic programming is [7], where extraction takes terminating processes typed using a fragment of linear logic as input. We advance both lines of work in several aspects, described below.

1.1 Contributions

Extraction for synchronous systems. We define an extraction procedure that applies directly to both choreography specifications and choreographic programming, by working with representative models. We focus on the more difficult case of choreographic programming, and then show how our approach can be applied to other settings in Sect. 6. First we define an extraction algorithm for processes with synchronous communications (Sect. 4), which showcases the key elements of our construction: building a choreography corresponds to finding paths in a graph that represents the abstract execution of the input processes. Our extraction also helps in debugging: if extraction detects a potential deadlock, we pinpoint it with a special term (**1**). This is the first extraction procedure for choreographic programming that can deal with procedures and infinite behavior [7].

Asynchrony. We extend our development to asynchronous communication (Sect. 5). The key novelty is that we can extract a new class of behaviors where processes progress because of asynchronous communication. The simplest example of this class is a two-way exchange: a network of two processes where each process starts by sending a value to the other, and then consumes the received value. This network is deadlocked under a synchronous semantics, violating the state-of-the-art requirements for extraction [19]. Capturing these behaviors is challenging for two reasons: there is no choreography language capable of representing them; and the extraction algorithms presented so far require the behaviors of processes to be representable also under a synchronous interpretation. We overcome both limitations with a new choreography primitive for multiparty asynchronous exchange and a look-ahead mechanism for asynchronous actions in extraction.

Efficiency. We show that our extraction has exponential worst-case time complexity in both the synchronous and the asynchronous cases (Sects. 4 and 5, respectively), unlike the factorial case of [19], even though we can capture a new class of behaviors. In particular, we need only one phase of exponential complexity, while [19] uses multiple phases applied in sequence. The authors of [19] detail only the complexities of their first two phases: the first has exponential complexity (but in a quantity larger than ours), while the second has factorial complexity in a function exponential in the size of the input. Our better time complexity stems from the design of our process language, which does not allow non-deterministic receives from different channels, and careful algorithm crafting. Despite the restriction, we can still model interesting examples thanks to asynchronous exchange. In Sect. 5, we present a novel formulation of the alternating 2-bit protocol, which is given in [15] and used in [19] as a motivating example. Our formulation is simpler and does not require threads as in [19].

2 Related Work

Choreographic Programming. The state of the art for extraction in choreographic programming is [7], where synchronous processes with finite behavior are typed using the multiplicative-additive fragment of linear logic. Our approach is significantly more expressive, bringing support for recursion and asynchronous communication. Also, the proof theory in [7] requires that there are no cycles in the structure of connections among processes. We do not have this limitation.

Choreography Specifications. To the best of our knowledge, the state of the art for extracting choreography specifications is [19], which captures more behaviors than previous works with similar objectives [18,22].

Extraction in [19] is more restrictive wrt. to asynchrony, requiring all process traces and choices to be represented in the synchronous transition system of the network. Thus, networks that are safe because of asynchronous communication are not extracted in [19]. Instead, our extraction can deal with programs that use multiparty asynchronous exchange, where multiple processes exchange values by exploiting asynchronous communication. As a consequence, we can extract the alternating 2-bit protocol implemented via asynchronous exchange in Sect. 5, which is deadlocked under a synchronous semantics and thus cannot be extracted in [19]. Our extraction is the first capturing systems that are not correctly approximated by synchronous semantics (cf. [2]). A precise characterization of the class of extractable systems is thus an interesting future direction.

To circumvent the limitation that asynchronous exchange is not supported, choreographies in [19] support local concurrency: processes can have internal threads. This opens up for an alternative formulation of the alternating 2-bit protocol, where the two participants use two threads each. However, these choreographies are harder to read. As an example, compare our choreography for the alternating 2-bit protocol in Sect. 5 to that obtained with the automata in [19] (given in [15], Protocol 7 in Example 2.1). Our formulation is a simple recursive

procedure with two exchanges, whereas the control flow in [15] is rather intricate and uses three different operators (fork, join, and merge) at different places to compose two separate loops. In our opinion, our choreographies follow the principles of structured programming to a greater extent, and are simpler; also because coordination happens only through communication.

More interestingly than readability, local concurrency makes the complexity of extraction blow up factorially [19]: process threads are represented using non-determinism between different actions in communicating automata. Determining whether the non-deterministic behavior of these automata is extractable takes (super-)factorial time (factorial time in the size of a graph similar to our AES, cf. Definition 2)! Thus, asynchronous exchange supports a more efficient way of capturing an interesting class of behaviors. Nevertheless, we believe that developing efficient extractions of local concurrency may be useful future work.

3 Core Choreographies and Stateful Processes

We review the languages of Core Choreographies (CC) and Stateful Processes (SP), from [11], which respectively model choreographies and endpoint programs. We introduce labels in the reduction semantics for these calculi to formalize the link between choreographies and their process implementations as a bisimilarity.

$$C ::= \mathbf{0} \mid \eta; C \mid \mathsf{if}\ \mathsf{p} \overset{\leftarrow}{=} \mathsf{q}\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2 \mid \mathsf{def}\ X = C_2\ \mathsf{in}\ C_1 \mid X$$
$$\eta ::= \mathsf{p}.e \texttt{ -> } \mathsf{q} \mid \mathsf{p} \texttt{ -> } \mathsf{q}[l] \qquad\qquad e ::= v \mid * \mid \ldots$$

Fig. 1. Core Choreographies, Syntax.

Core Choreographies (CC). The syntax of CC is given in Fig. 1. A choreography C describes the behavior of a set of processes (p, q, ...) running concurrently. Each process has an internal memory cell storing a local value (the value of the process). Term $\mathbf{0}$ is the terminated choreography (omitted in examples). Term $\eta; C$ reads "the system executes η and proceeds as C". An interaction η is either: a value communication $\mathsf{p}.e$ -> q, where process p evaluates e and sends the result to process q, which stores it in its memory cell, replacing its previous value; or a selection p -> $\mathsf{q}[l]$, where p selects l among the branches offered by q. We abstract from the concrete language of expressions e, which models internal computation and is orthogonal to our development, assuming only that: expressions can contain values v and the placeholder $*$, which refers to the value of the process evaluating them; and evaluating expressions always terminates and returns a value. In a conditional $\mathsf{if}\ \mathsf{p} \overset{\leftarrow}{=} \mathsf{q}\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2$, p checks if its value is equal to q's to decide whether the system proceeds as C_1 or C_2. Term $\mathsf{def}\ X = C_2\ \mathsf{in}\ C_1$ defines a procedure X with body C_2, which can be called in C_1 and C_2 by using term X.

$$\frac{e[\sigma(\mathsf{p})/*] \downarrow v}{\mathsf{p}.e \texttt{ -> } \mathsf{q}; C, \sigma \xrightarrow{\mathsf{p}.v \texttt{ -> } \mathsf{q}} C, \sigma[\mathsf{q} \mapsto v]} \lfloor \text{C|Com} \rceil \qquad \frac{}{\mathsf{p} \texttt{ -> } \mathsf{q}[l]; C, \sigma \xrightarrow{\mathsf{p} \texttt{ -> } \mathsf{q}[l]} C, \sigma} \lfloor \text{C|Sel} \rceil$$

$$\frac{\sigma(\mathsf{p}) = \sigma(\mathsf{q})}{\mathsf{if}\ \mathsf{p} \overset{\leftarrow}{=} \mathsf{q}\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2, \sigma \xrightarrow{\mathsf{p} \overset{\leftarrow}{=} \mathsf{q}:\mathsf{then}} C_1, \sigma} \lfloor \text{C|Then} \rceil \qquad \frac{\mathsf{pn}(\eta) \cap \mathsf{pn}(\eta') = \emptyset}{\eta; \eta' \preceq \eta'; \eta} \lfloor \text{C|Eta-Eta} \rceil$$

Fig. 2. Core choreographies, semantics and structural precongruence (selected rules).

The semantics of CC is given in terms of labeled reductions $C, \sigma \xrightarrow{\lambda} C', \sigma'$; the main reduction rules are given in Fig. 2. Reductions are also closed under context (procedure definitions) and under a structural precongruence $\preceq$, allowing procedure calls to be unfolded and non-interfering actions to be executed in any order. The most interesting rule for $\preceq$ is rule $\lfloor$C|Eta-Eta$\rceil$, which swaps communications between disjoint sets of processes (modeling concurrency). The total function σ maps each process name to the value it stores. Labels λ tell us which action has been performed, which helps stating our later results. In rule $\lfloor$C|Com$\rceil$, v is the value obtained by evaluating ($\downarrow$) the expression e, with $*$ replaced by the value of the sender p, $\sigma(\mathsf{p})$. In the reductum, σ is updated such that the receiver q stores v. Rule $\lfloor$C|Sel$\rceil$ does not alter σ: selections model invoking a method/operation available at the receiver. Rules $\lfloor$C|Then$\rceil$ and $\lfloor$C|Else$\rceil$ (omitted) model conditionals in the standard way. Function $\mathsf{pn}(C)$ returns all the process names that appear in C, and $C \equiv C'$ means $C \preceq C'$ and $C' \preceq C$.

Example 1. We define a simple choreography for client authentication. We write p -> $\mathsf{c}, \mathsf{s}[l]$ as a shortcut for p -> $\mathsf{c}[l]; \mathsf{p}$ -> $\mathsf{s}[l]$.

$$\mathsf{def}\ X = \Big(\mathsf{c}.pwd \texttt{ -> } \mathsf{a}; \mathsf{if}\ \mathsf{a} \overset{\leftarrow}{=} \mathsf{s}\ \mathsf{then}\ (\mathsf{a} \texttt{ -> } \mathsf{c}, \mathsf{s}[ok]; \mathsf{s}.t \texttt{ -> } \mathsf{c})\ \mathsf{else}\ (\mathsf{a} \texttt{ -> } \mathsf{c}, \mathsf{s}[ko]; X)\Big)\ \mathsf{in}\ X$$

In this choreography, a client process c sends a password to an authentication process a, which checks if the password matches that contained in the server-side process s. If the password is correct, a notifies c and s, and s sends an authentication token t to c. Otherwise, a notifies c and s that authentication failed, and a new attempt is made (by recursively invoking X). □

Stateful Processes. The calculus SP models concurrent/distributed implementations. Thus, unlike in CC, actions are now distributed among processes.

The syntax of SP is given in Fig. 3. Networks N are parallel compositions of processes $\mathsf{p} \triangleright B$, read "process p has behavior B". An output term $\mathsf{q}!\langle e \rangle; B$ sends the result of evaluating e to q, and then proceeds as B. Outputs are meant to

$$B ::= \mathsf{q}!\langle e \rangle; B \mid \mathsf{p}?; B \mid \mathsf{q} \oplus l; B \mid \mathsf{p}\&\{l_i : B_i\}_{i \in I} \mid \mathbf{0} \mid \mathsf{if} * \overset{\leftarrow}{=} \mathsf{q}\ \mathsf{then}\ B_1\ \mathsf{else}\ B_2 \mid \mathsf{def}\ X = B_2\ \mathsf{in}\ B_1 \mid X$$

$$N ::= \mathsf{p} \triangleright B \mid \mathbf{0} \mid N \mid N$$

Fig. 3. Stateful processes, syntax.

$$\frac{e[\sigma(\mathsf{p})/*] \downarrow v}{\mathsf{p} \triangleright \mathsf{q}!\langle e\rangle; B_1 \mid \mathsf{q} \triangleright \mathsf{p}?; B_2,\ \sigma \xrightarrow{\mathsf{p}.v\ \text{->}\ \mathsf{q}} \mathsf{p} \triangleright B_1 \mid \mathsf{q} \triangleright B_2,\ \sigma[\mathsf{q} \mapsto v]} \lfloor \text{S}|\text{Com} \rceil$$

$$\frac{j \in I}{\mathsf{p} \triangleright \mathsf{q} \oplus l_j; B \mid \mathsf{q} \triangleright \mathsf{p}\&\{l_i : B_i\}_{i\in I},\ \sigma \xrightarrow{\mathsf{p}\ \text{->}\ \mathsf{q}[l]} \mathsf{p} \triangleright B \mid \mathsf{q} \triangleright B_j,\ \sigma} \lfloor \text{S}|\text{Sel} \rceil$$

$$\frac{e[\sigma(\mathsf{q})/*] \downarrow \sigma(\mathsf{p})}{\mathsf{p} \triangleright \mathsf{if} * \overset{\leftarrow}{=} \mathsf{q}\ \mathsf{then}\ B_1\ \mathsf{else}\ B_2 \mid \mathsf{q} \triangleright \mathsf{p}!\langle e\rangle; B',\ \sigma \xrightarrow{\mathsf{p}\overset{\leftarrow}{=}\mathsf{q}:\mathsf{then}} \mathsf{p} \triangleright B_1 \mid \mathsf{q} \triangleright B',\ \sigma} \lfloor \text{S}|\text{Then} \rceil$$

Fig. 4. Stateful processes, semantics (selected rules).

synchronize with input terms at the target process, i.e., $\mathsf{p}?; B$, which receives a value from p to be stored locally and then proceeds as B. Term $\mathsf{q} \oplus l; B$ sends the selection of the branch labeled l to q. Branches are offered by the receiver with term $\mathsf{p}\&\{l_i : B_i\}_{i\in I}$, which offers a choice among the labels l_i to p. When one of these labels is selected, the respective behavior B_i is run. Term $\mathsf{if} * \overset{\leftarrow}{=} \mathsf{q}\ \mathsf{then}\ B_1\ \mathsf{else}\ B_2$ communicates with process q to check whether it stores the same value as the process running this behavior, in order to choose between the continuations B_1 and B_2. Terms $\mathsf{def}\ X = B_2\ \mathsf{in}\ B_1$ and X are procedure definition and call, respectively.

The semantics of SP is given by labeled reductions $N, \sigma \xrightarrow{\lambda} N', \sigma'$, with labels λ as in CC.[1] Figure 4 shows the key rules (see the appendix for the complete set). Two processes can synchronize when they refer to each other. In rule $\lfloor$S|Com$\rceil$, an output at p directed at q synchronizes with the dual input action at q – intention to receive from p; in the reductum, q's value is updated. The reduction receives the same label as the equivalent communication term in CC. The other rules shown are similar. The omitted rules are standard, and close the semantics under parallel composition, structural precongruence, and procedure definitions.

Example 2. The following network implements the choreography in Example 1.

$$\begin{aligned}
&\mathsf{c} \triangleright \mathsf{def}\ X = \mathsf{a}!\langle pwd\rangle; \mathsf{a}\&\{ok : \mathsf{s}?,\ ko : X\}\ \mathsf{in}\ X \\
\mid\ &\mathsf{a} \triangleright \mathsf{def}\ X = \mathsf{c}?; \mathsf{if} * \overset{\leftarrow}{=} \mathsf{s}\ \mathsf{then}\ (\mathsf{c} \oplus ok; \mathsf{s} \oplus ok)\ \mathsf{else}\ (\mathsf{c} \oplus ko; \mathsf{s} \oplus ko; X)\ \ \mathsf{in}\ X \\
\mid\ &\mathsf{s} \triangleright \mathsf{def}\ X = \mathsf{a}!\langle *\rangle; \mathsf{a}\&\{ok : \mathsf{c}!\langle t\rangle,\ ko : X\}\ \mathsf{in}\ X
\end{aligned}$$

EndPoint Projection (EPP). As shown in [11], there exists a partial function $[\![\cdot]\!] : \text{CC} \to \text{SP}$, called EndPoint Projection (EPP), that produces correct implementations of choreographies. EPP produces a parallel composition of processes, one for each process name in the original choreography: $[\![C]\!] = \prod_{\mathsf{p}\in \mathsf{pn}(C)} \mathsf{p} \triangleright [\![C]\!]_{\mathsf{p}}$. The rules for computing $[\![C]\!]$ project the local action performed by the process of interest. For example, $[\![\mathsf{p}.e\ \text{->}\ \mathsf{q}]\!]_{\mathsf{p}} = \mathsf{q}!\langle e\rangle$ and $[\![\mathsf{p}.e\ \text{->}\ \mathsf{q}]\!]_{\mathsf{q}} = \mathsf{p}?$.

[1] Deviating from [11], we model process values using σ as for CC, for simplicity.

The network presented in Example 2 is exactly the EPP of the choreography in Example 1. Observe that the projection of the conditional in the original choreography for the processes c and s is a branching that supports all the possible choices made by process a in its projected conditional. Producing these branching terms is possible only if, whenever there is a conditional at a process (a in our example), all other processes receive a label that tells them which branch such a process has chosen. (In case the behaviors of the other processes are the same in both cases, producing branching terms is not necessary.) When this cannot be done for a choreography C, the EPP for C is undefined, and we say that C is unprojectable. Conversely, C is *projectable* if $[\![C]\!]$ is defined.

In the remainder, we relate choreographies to network implementations via a strong labeled reduction bisimilarity $\sim$. Bisimilarity is defined as usual [24]: it is the union of all bisimulation relations $\mathcal{R}$, which in our case relate choreographies to networks. A relation $\mathcal{R}$ is one such bisimulation if whenever $C\mathcal{R}N$ we have that, for all σ: (i) $C, \sigma \xrightarrow{\lambda} C', \sigma'$ implies $N, \sigma \xrightarrow{\lambda} N', \sigma'$ for some N' with $C'\mathcal{R}N'$; (ii) $N, \sigma \xrightarrow{\lambda} N', \sigma'$ implies $C, \sigma \xrightarrow{\lambda} C', \sigma'$ for some C' with $C'\mathcal{R}N'$.

Theorem 1 (adapted from [11]). *If C is projectable, then $C \sim [\![C]\!]$.*

4 Extraction from SP

The finite case. We first investigate *finite SP*, the fragment of SP without recursive definitions, which we use to discuss the intuition behind our extraction.

Definition 1. *We define a rewriting relation $\rightsquigarrow$ on the language of CC extended with terms $([\![N]\!])$, where N is a network in finite SP, as the transitive closure of:*

$$\frac{N \equiv \mathbf{0}}{([\![N]\!]) \rightsquigarrow \mathbf{0}} \qquad \frac{N \equiv \mathsf{p} \triangleright \mathsf{q}!\langle e\rangle; N_p \mid \mathsf{q} \triangleright \mathsf{p}?; N_q \mid N'}{([\![N]\!]) \rightsquigarrow \mathsf{p}.e \to \mathsf{q}; ([\![N_p \mid N_q \mid N']\!])}$$

$$\frac{N \equiv \mathsf{p} \triangleright \mathsf{q} \oplus l_k; N_p \mid \mathsf{q} \triangleright \mathsf{p}\&\{l_1 : N_{q_1}, \ldots, l_n : N_{q_n}\} \mid N'}{([\![N]\!]) \rightsquigarrow \mathsf{p} \to \mathsf{q}[l_k]; ([\![N_p \mid N_{q_k} \mid N']\!])}$$

$$\frac{N \equiv \mathsf{p} \triangleright \mathsf{if} * \overset{\leftarrow}{=} \mathsf{q} \;\mathsf{then}\; N_{p_1} \;\mathsf{else}\; N_{p_2} \mid \mathsf{q} \triangleright \mathsf{p}!\langle e\rangle; N_q \mid N'}{([\![N]\!]) \rightsquigarrow \mathsf{if}\; \mathsf{p} \overset{\leftarrow}{=} \mathsf{q} \;\mathsf{then}\; ([\![N_{p_1} \mid N_q \mid N']\!]) \;\mathsf{else}\; ([\![N_{p_2} \mid N_q \mid N']\!])} \qquad \frac{\textit{no other rule applies}}{([\![N]\!]) \rightsquigarrow \mathbf{1}}$$

A network N in finite SP extracts *to a choreography C if $([\![N]\!]) \rightsquigarrow C$.*

The last rule guarantees that every network is extractable. Extraction uses structural precongruence (namely, commutativity and associativity of parallel composition) to find matching actions. For finite SP, this is not a problem (the set of networks equivalent to a given one is finite), but it makes extraction nondeterministic, e.g., the network $\mathsf{p} \triangleright \mathsf{q}!\langle e\rangle \mid \mathsf{q} \triangleright \mathsf{p}? \mid \mathsf{r} \triangleright \mathsf{s}!\langle e'\rangle \mid \mathsf{s} \triangleright \mathsf{r}?$ extracts both to $\mathsf{p}.e \to \mathsf{q}; \mathsf{r}.e' \to \mathsf{s}$ and $\mathsf{r}.e' \to \mathsf{s}; \mathsf{p}.e \to \mathsf{q}$. These choreographies are equivalent by Rule $\lfloor$C|Eta-Eta$\rceil$ (Fig. 2). This holds in general, as stated below.

Lemma 1. *If $([\![N]\!]) \rightsquigarrow C_1$ and $([\![N]\!]) \rightsquigarrow C_2$, then $C_1 \equiv C_2$.*

$$\frac{}{\mathsf{p} \triangleright \mathsf{q}!\langle e\rangle; B_1 \mid \mathsf{q} \triangleright \mathsf{p}?; B_2 \xrightarrow{\mathsf{p}.e \to \mathsf{q}} \mathsf{p} \triangleright B_1 \mid \mathsf{q} \triangleright B_2} \lfloor \mathrm{S|Com} \rceil$$

$$\frac{}{\mathsf{p} \triangleright \mathsf{if} * \overset{\leftarrow}{=} \mathsf{q} \mathsf{\ then\ } B_1 \mathsf{\ else\ } B_2 \mid \mathsf{q} \triangleright \mathsf{p}!\langle e\rangle; B' \xrightarrow{\mathsf{p} \overset{\leftarrow}{=} \mathsf{q}:\mathsf{then}} \mathsf{p} \triangleright B_1 \mid \mathsf{q} \triangleright B'} \lfloor \mathrm{S|Then} \rceil$$

Fig. 5. Stateful Processes, Abstract Semantics (selected rules).

There is one important design option to consider: what to do with actions that cannot be matched, i.e., processes that will deadlock. There are two alternatives: restrict extraction to lock-free networks (networks where all processes eventually progress, in the sense of [8]); or extract stuck processes to a new choreography term **1**, with the same semantics as **0**. We choose the latter option for debugging reasons. Specifically, practical applications of extraction may annotate **1** with the code of the deadlocked processes, giving the programmer a chance to see exactly where the system is unsafe, and attempt at fixing it manually. Better yet: since the code to unlock deadlocked processes in process calculi can be efficiently synthesized [8], our method may be integrated with the technique in [8] to suggest an automatic system repair.

Remark 1. If $(\![N]\!) \rightsquigarrow C$ and C does not contain **1**, then N is lock-free. However, even if C contains **1**, N may still be lock-free: the code causing the deadlock may be dead code in a conditional branch that is never chosen during execution.

Extraction is sound: it yields a choreography that is bisimilar to the original network. Also, for finite SP, it behaves as an inverse of EPP.

Theorem 2. *Let N be in finite SP. If $(\![N]\!) \rightsquigarrow C$, then $C \sim N$. Furthermore, if $N = [\![C']\!]$ for some C', then $(\![N]\!) \rightsquigarrow C'$.*

As we show later, the second part of this theorem does not hold in the presence of recursive definitions.

We now restate extraction in terms of a particular graph, which is the hallmark of our development: when we add recursion to SP, we can no longer define extraction as a set of rewriting rules. We first introduce a new abstract semantics for networks, $N \xrightarrow{\alpha} N'$, defined as in Fig. 4 except for the rules for value communication and conditionals, which are replaced by those in Fig. 5 (we omit the obvious rule $\lfloor \mathrm{S|Else} \rceil$). In particular, conditionals are now nondeterministic. Labels α are like λ but may now contain expressions (see the new rule $\lfloor \mathrm{S|Com} \rceil$); in all other rules, λ is replaced by α. We write $N \xrightarrow{\tilde{\alpha}}{}^{*} N'$ for $N \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} N'$.

Definition 2. *Given a network N, the* Abstract Execution Space (AES) *of N is the directed graph obtained by considering all possible abstract reduction paths from N. Its vertices are all the networks N' such that $N \xrightarrow{\tilde{\alpha}}{}^{*} N'$, and there is an edge between two vertices N_1 and N_2 labeled α if $N_1 \xrightarrow{\alpha} N_2$.*

A Symbolic Execution Graph (SEG) *for N is a subgraph of its AES that contains N and such that each vertex $N' \not\equiv \mathbf{0}$ has either one outgoing edge labeled by an η or two outgoing edges labeled $\mathsf{p} \overset{\leftarrow}{=} \mathsf{q} : \mathsf{then}$ and $\mathsf{p} \overset{\leftarrow}{=} \mathsf{q} : \mathsf{else}$.*

Intuitively, the AES of N represents all possible evolutions of N (each evolution is a path in this graph). A SEG fixes the order of execution of actions, but still abstracts from the state (and thus considers both branches of conditionals). For networks in finite SP, these graphs are finite.

Given a network N, there is a one-to-one correspondence between SEGs for N and choreographies C such that $(\![N]\!) \rightsquigarrow C$. Indeed, given a SEG we can extract a choreography as follows. We start from the initial vertex, labeled N. If there is an outgoing edge with label η to N', we add η to the choreography and continue from N'. If there are two outgoing edges with labels $\mathsf{p} \overset{\leftarrow}{=} \mathsf{q} : \mathsf{then}$ and $\mathsf{p} \overset{\leftarrow}{=} \mathsf{q} : \mathsf{else}$ to N_1 and N_2, respectively, we extract a conditional whose branches are the choreographies extracted by continuing exploration from N_1 and N_2, respectively. When we reach a leaf, we extract $\mathbf{0}$ or $\mathbf{1}$, according to whether its label is equivalent to $\mathbf{0}$ or not. Conversely, we can build a SEG from a particular rewriting of $(\![N]\!)$ by following the choreography actions one at a time.

Treating recursive definitions. We now extend extraction to networks with recursive definitions, using SEGs. We need to be careful with the definition of the AES, since including all possible (abstract) executions now may make it infinite (due to recursion unfolding), and thus extraction may not terminate. To avoid this, we only allow recursive definitions to be unfolded (once) if they occur at the head of a process involved in a reduction. With this restriction, we can define the AES and SEGs for a network as in the finite case. These graphs may now contain cycles: a network may evolve into the same term after a few reductions.

Example 3. Consider the following network.

$$\mathsf{p} \triangleright \mathsf{def}\ X = \mathsf{q}!\langle * \rangle; \mathsf{q}\&\{\mathrm{L} : \mathsf{q}!\langle * \rangle; X, \mathrm{R} : \mathbf{0}\}\ \mathsf{in}\ \mathsf{q}!\langle * \rangle; X$$
$$\mid \mathsf{q} \triangleright \mathsf{def}\ Y = \mathsf{p}?; \mathsf{p}?; \mathsf{if} * \overset{\leftarrow}{=} \mathsf{r}\ \mathsf{then}\ \mathsf{p} \oplus \mathrm{L}; Y\ \mathsf{else}\ \mathsf{p} \oplus \mathrm{R}; \mathbf{0}\ \mathsf{in}\ Y \mid \mathsf{r} \triangleright \mathsf{def}\ Z = \mathsf{q}!\langle * \rangle; Z\ \mathsf{in}\ Z$$

This network generates the AES in Fig. 6, which is also its SEG. □

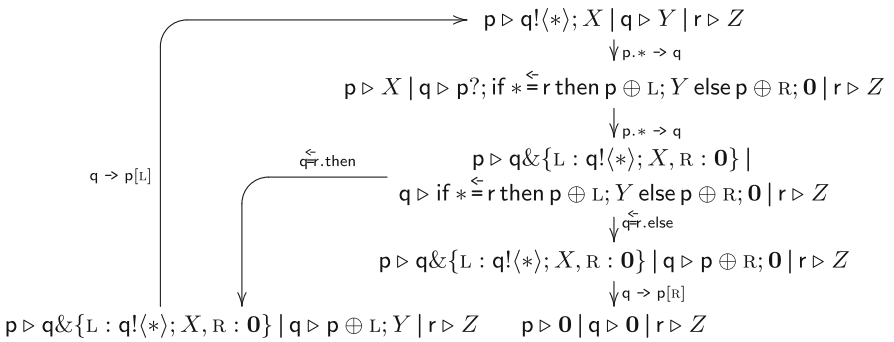


Fig. 6. The AES and SEG for the network in Example 3.

The key insight is that the definitions of recursive procedures are extracted from the loops in the SEG, rather than from the recursive definitions in the source network. This construction typically yields mutually recursive definitions, motivating a small change to CC that does not add expressivity: we replace the constructor $\mathsf{def}\ X = C_2\ \mathsf{in}\ C_1$ by top-level procedure definitions, in the style of [12]. A choreography now becomes a pair $\langle \mathcal{D}, C \rangle$, where $\mathcal{D} = \{X_i = C_i\}$ and all procedure calls in either C or the C_i are to some X_i defined in D.

Definition 3. *The choreography extracted from a SEG is defined as follows. We annotate each node that has more than one incoming edge with a unique procedure identifier. Then, for every node annotated with an identifier, say X, we replace each of its incoming edges with an edge to a new leaf node that contains a special term X (so now the node annotated with X has no incoming edges). This eliminates all loops in the SEG, allowing us to reuse the extraction procedure for the non-recursive case to extract the desired pair $\langle \mathcal{D}, C \rangle$. We get C by extraction starting from the initial network. Then, for each node that we annotated with an X, we extract a choreographic procedure X in $\mathcal{D}$ that has as body the choreography extracted from the graph that starts from that annotated node. Any new leaf node containing a special term X is extracted as a procedure call X.*

Example 4. Consider the SEG in Fig. 3. To extract a choreography, we annotate the topmost node with a procedure identifier X and replace the incoming edge to that node with an edge to a new leaf X. We thus extract X to be

$$\mathsf{p}.*\mathsf{q};\ \mathsf{p}.* \rightarrow \mathsf{q};\ \mathsf{if}\ \mathsf{q} \overset{<}{=} \mathsf{r}\ \mathsf{then}\ \mathsf{q} \rightarrow \mathsf{p}[\mathrm{L}];\ X\ \mathsf{else}\ \mathsf{q} \rightarrow \mathsf{p}[\mathrm{R}];\ \mathbf{1}$$

and the extracted choreography itself is simply X. The body of X is not projectable (the branches for r are not mergeable, cf. [11]), but it faithfully describes the behavior of the original network. □

The procedure in Definition 3 always terminates, but sometimes it yields choreographies that starve some processes. As an example, the network

$$\begin{aligned} &\mathsf{p} \triangleright \mathsf{def}\ X = \mathsf{q}!\langle * \rangle; X\ \mathsf{in}\ X \ \mid\ \mathsf{q} \triangleright \mathsf{def}\ Y = \mathsf{p}?; Y\ \mathsf{in}\ Y \\ &\mid \mathsf{r} \triangleright \mathsf{def}\ Z = \mathsf{s}!\langle * \rangle; Z\ \mathsf{in}\ Z \ \mid\ \mathsf{s} \triangleright \mathsf{def}\ W = \mathsf{r}?; W\ \mathsf{in}\ W \end{aligned} \tag{1}$$

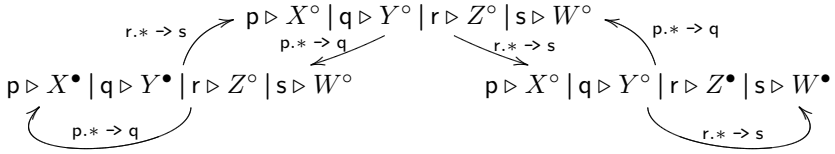
has two SEGs, which extract to the choreographies $\mathsf{def}\ X = \mathsf{p}.* \rightarrow \mathsf{q}; X\ \mathsf{in}\ X$ and $\mathsf{def}\ X = \mathsf{r}.* \rightarrow \mathsf{s}; X\ \mathsf{in}\ X$, none of which captures all the behaviors of N.

To avoid this problem, we change the definitions of AES and SEGs slightly. We annotate all procedure calls in networks with either ∘ or •. The node in the AES corresponding to the initial network has all procedure calls annotated with ∘. There is an edge from N to N' with label α if $N \xrightarrow{\alpha} N'$ and the procedure calls in N' are annotated as follows.

- If executing α does not require unfolding procedure calls, then all calls in N' are annotated as in N.
- If executing α requires unfolding procedure calls, then we annotate all the calls in N' introduced by these unfoldings with •. If N' now has *all* procedure calls annotated with •, we change all annotations to ∘.

We then require loops in a SEG to contain a node where every procedure call is annotated with $\circ$. This ensures that every procedure call is unfolded at least once before returning to the same node. This holds even if $\mathsf{p} \triangleright X$ unfolds to a behavior that calls different procedures, but not X: in order to return to the same node, the newly invoked procedures themselves need to be unfolded.

Example 5. The annotated AES for the network (1) is:



This AES now has the following two SEGs:

Observe that the self-loops are discarded because they do not go through a node with all $\circ$ annotations. From these SEGs, we can extract two definitions for X:

$$\mathsf{def}\ X = \mathsf{p}.* \texttt{->} \mathsf{q}; \mathsf{r}.* \texttt{->} \mathsf{s}; X\ \mathsf{in}\ X \qquad \text{and} \qquad \mathsf{def}\ X = \mathsf{r}.* \texttt{->} \mathsf{s}; \mathsf{p}.* \texttt{->} \mathsf{q}; X\ \mathsf{in}\ X$$

Both of these definitions correctly capture all behaviors of the network. □

A similar situation may occur if there are processes with finite behavior (no procedure calls): the network

$$\mathsf{p} \triangleright \mathsf{def}\ X = \mathsf{q}!\langle * \rangle; X\ \mathsf{in}\ X \ \mid\ \mathsf{q} \triangleright \mathsf{def}\ Y = \mathsf{p}?; Y\ \mathsf{in}\ Y \ \mid\ \mathsf{r} \triangleright \mathsf{s}!\langle * \rangle \ \mid\ \mathsf{s} \triangleright \mathsf{r}?$$

can be extracted to the choreography X, with $X = \mathsf{p}.* \texttt{->} \mathsf{q}; X$, where r and s never communicate. Hence, we require that if a node in a SEG has more than one incoming edge (it is a "loop" node) and contains processes with finite behavior, then these processes must be deadlocked (being finite, this is trivially verifiable). This ensures that if finite processes are able to reduce, they cannot be in a loop.

Definition 4. *A SEG for a network N is valid if all its loops:*

- *pass through a node where all recursive calls are marked with $\circ$;*
- *start in a node where all processes with finite behavior are deadlocked.*

A network N extracts to a choreography C if C can be constructed (as in Definition 3) from a valid SEG for N.

Validity implies, however, that there are some non-deadlocked networks that are not extractable, such as

$$\mathsf{p} \triangleright \mathsf{def}\ X = \mathsf{q}!\langle * \rangle; X\ \mathsf{in}\ X \mid \mathsf{q} \triangleright \mathsf{def}\ Y = \mathsf{p}?; Y\ \mathsf{in}\ Y \mid \mathsf{r} \triangleright \mathsf{def}\ Z = \mathsf{p}?; Z\ \mathsf{in}\ Z$$

for which there is no valid SEG. This is to be expected, since deadlock-freedom is undecidable in SP. We can generalize this observation as a necessary condition for extraction to be defined, in the following theorem.

Theorem 3. *If the AES for a network N does not contain nodes from which a process is always deadlocked, then N is extractable.*

Lemma 1 and the first part of Theorem 2 still hold for extraction in SP with recursion, but the second part of Theorem 2 does not: in general, the projection of a choreography is extracted to a choreography with different procedures, since extraction ignores the actual definitions in the source network.

Theorem 4. *If C is a choreography extracted from a network N, then $N \sim C$.*

We conclude this section with some complexity theoretical considerations.

Lemma 2. *The annotated AES for a network of size n has at most $e^{\frac{2n}{e}}$ vertices.*

Theorem 5. *Extraction from a network of size n terminates in time $O(ne^{\frac{2n}{e}})$.*

As discussed earlier, this time complexity is a dramatic improvement over earlier, comparable work. However, in practice, we may be able to perform even better. Algorithmically, all the required work stems from traversals of the AES, so any reduction in its (explored) size will lead to proportional runtime improvements. Thus, instead of first computing the entire AES and then a valid SEG, we can compute the relevant parts of the AES lazily as we need them, so parts of the AES that are never explored while computing a valid SEG are never generated.

5 Asynchrony

We now discuss an asynchronous semantics for SP, with which we can express new safe behaviors. Most notably, SP can now express asynchronous exchange (Example 6). We also show a novel choreography primitive that successfully captures this pattern, which cannot be described in previous works on choreographic programming, and extend our algorithm to extract it from networks.

Asynchronous SP. Asynchronous communication can be added to SP using standard techniques for process calculi. In the semantics of networks, we add a FIFO queue for each pair of processes. Communications now synchronize with these queues: send actions append a message in the queue of the receiver, and receive actions remove the first message from the queue of the receiver (see [12] for a formalization in an extension of SP).

Example 6. The network $\mathsf{p} \triangleright \mathsf{q}!\langle * \rangle; \mathsf{q}? \mid \mathsf{q} \triangleright \mathsf{p}!\langle * \rangle; \mathsf{p}?$ exemplifies the pattern of asynchronous exchange. This network is deadlocked in synchronous SP, but runs without errors in asynchronous SP: both p and q can send their respective values, becoming ready to receive each other's messages. This behavior is not representable in any previous work on choreographies (including CC from Sect. 3), since all choreographies presented so far can only describe processes that are not deadlocked under a synchronous semantics (see [12] for a formal argument). □

The multicom. The situation in Example 6 is prototypical of programs that are safe only in an asynchronous setting: a group of processes wants to send messages to a group of receivers, with circular dependencies among communications.

We deal with this situation by means of a new choreography action, which we call a *multicom*. Syntactically, a multicom is a list of communication actions with distinct receivers, which we write $(\tilde{\eta})$. In the unary case, we obtain the usual communications and selections; by removing these from the syntax of CC and adding the multicom, we obtain a more expressive calculus with fewer primitives. The semantics of multicom is given by the following rule, which generalizes (and replaces) both ⌊C|Com⌉ and ⌊C|Sel⌉.

$$\frac{I = \{i \mid \mathsf{p}_i.e_i \texttt{ -> } \mathsf{q}_i \in \tilde{\eta}\} \qquad v_i = e_i[\sigma(\mathsf{p}_i)/*]}{(\tilde{\eta}); C, \sigma \xrightarrow{(\tilde{\eta})[e_i/v_i]_{i\in I}} C, \sigma[\mathsf{q}_i \mapsto v_i]_{i\in I}} \lfloor \text{C|MCom} \rceil$$

Structural precongruence rules for the multicom are motivated by its intuitive semantics: actions inside a multicom can be permuted as long as the senders differ, and sequential multicoms can be merged as long as they do not share receivers and there are no sequential constraints between them (i.e., none of the receivers in the first multicom is a sender in the second one).

$$\frac{\mathsf{pn}(\eta_1) \cap \mathsf{pn}(\eta_2) = \emptyset}{\big(\ldots, \eta_1, \eta_2, \ldots\big) \equiv \big(\ldots, \eta_2, \eta_1, \ldots\big)} \lfloor \text{C|MCom-Perm} \rceil$$

$$\frac{\mathsf{rcv}(\eta) \cap \mathsf{rcv}(\nu) = \emptyset \qquad \mathsf{rcv}(\tilde{\eta}) \cap \mathsf{snd}(\tilde{\nu}) = \emptyset}{\big(\tilde{\eta}\big)\,;\big(\tilde{\nu}\big) \equiv \big(\tilde{\eta}, \tilde{\nu}\big)} \lfloor \text{C|MCom-MCom} \rceil$$

From these rules we can derive all instances of ⌊C|Eta-Eta⌉, e.g.:

$$\mathsf{p}.* \texttt{ -> } \mathsf{q}; \mathsf{r}.* \texttt{ -> } \mathsf{s} \equiv \begin{pmatrix} \mathsf{p}.* \texttt{ -> } \mathsf{q} \\ \mathsf{r}.* \texttt{ -> } \mathsf{s} \end{pmatrix} \equiv \begin{pmatrix} \mathsf{r}.* \texttt{ -> } \mathsf{s} \\ \mathsf{p}.* \texttt{ -> } \mathsf{q} \end{pmatrix} \equiv \mathsf{r}.* \texttt{ -> } \mathsf{s}; \mathsf{p}.* \texttt{ -> } \mathsf{q}$$

The problematic program in Example 6 can now be written as $\begin{pmatrix} \mathsf{p}.* \texttt{ -> } \mathsf{q} \\ \mathsf{q}.* \texttt{ -> } \mathsf{p} \end{pmatrix}$.

Structural precongruence rules for multicom also allow us to define a normal form for choreographies, where no multicom can be split in smaller multicoms.

Extraction. In order to extract choreographies containing multicoms, we alter the definition of the AES for a process network by allowing multicoms as labels for the edges. These can be computed using the following iterative algorithm.

1. For a process p with behavior $\mathsf{q}!\langle e\rangle; B$ (or $\mathsf{q} \oplus l; B$), set $\mathsf{actions} = \emptyset$ and $\mathsf{waiting} = \{\mathsf{p}.e \texttt{ -> } \mathsf{q}\}$ (resp. $\mathsf{waiting} = \{\mathsf{p} \texttt{ -> } \mathsf{q}[l]\}$).
2. While $\mathsf{waiting} \neq \emptyset$:
   (a) Move an action η from waiting to actions. Assume η is of the form $\mathsf{r}.e \texttt{ -> } \mathsf{s}$ (the case for label selection is similar).

(b) If the behavior of s is of the form $a_1; \ldots; a_k; \mathsf{r}?; B$ where each a_i is either the sending of a value or a label selection, then: for each a_i, if the corresponding choreography action is not in actions, add it to waiting.

3. Return actions.

This algorithm may fail (the behavior of s in step 2(b) is not of the required form), in which case the action initially chosen cannot be unblocked by a multicom.

Example 7. Consider the network from Example 6. Starting with action $\mathsf{q}!\langle * \rangle$ at process p, we initialize actions $= \emptyset$ and waiting $= \{\mathsf{p}.* \to \mathsf{q}\}$. We pick the action $\mathsf{p}.* \to \mathsf{q}$ from waiting and move it to actions. The behavior of q is $\mathsf{p}!\langle * \rangle; \mathsf{p}?$, which is of the form described in step 2(b); the choreography action corresponding to $\mathsf{p}!\langle * \rangle$ is $\mathsf{q}.* \to \mathsf{p}$, so we add this action to waiting, obtaining actions $= \{\mathsf{p}.* \to \mathsf{q}\}$ and waiting $= \{\mathsf{q}.* \to \mathsf{p}\}$. Now we consider the action $\mathsf{q}.* \to \mathsf{p}$, which we move from waiting to action, and look at p's behavior, which is $\mathsf{q}!\langle * \rangle; \mathsf{q}?$. The choreography action corresponding to $\mathsf{q}!\langle * \rangle$ is $\mathsf{p}.* \to \mathsf{q}$, which is already in actions, so we do not change waiting. The set waiting is now empty, and the algorithm terminates, returning $\begin{pmatrix} \mathsf{p}.* \to \mathsf{q} \\ \mathsf{q}.* \to \mathsf{p} \end{pmatrix}$. We would obtain the equivalent $\begin{pmatrix} \mathsf{q}.* \to \mathsf{p} \\ \mathsf{p}.* \to \mathsf{q} \end{pmatrix}$ by starting with the send action at q. □

Example 8. As a more sophisticated example, we show how our new choreographies with multicom can model the alternating 2-bit protocol. Here, Alice alternates between sending a 0 and a 1 to Bob; in turn, Bob sends an acknowledgment for every bit he receives, and Alice waits for the acknowledgment before sending another copy of the same bit. Since we are in an asynchronous semantics, we only consider the time when the messages arrive. With this in mind, we can write this protocol as the following network.

$$\begin{aligned} &\mathsf{a} \triangleright \mathsf{def}\ X = (\mathsf{b}?; \mathsf{b}!\langle 0 \rangle; \mathsf{b}?; \mathsf{b}!\langle 1 \rangle; X)\ \mathsf{in}\ (\mathsf{b}!\langle 0 \rangle; \mathsf{b}!\langle 1 \rangle; X) \\ \mid\ &\mathsf{b} \triangleright \mathsf{def}\ Y = (\mathsf{a}?; \mathsf{a}!\langle \mathsf{ack}_0 \rangle; \mathsf{a}?; \mathsf{a}!\langle \mathsf{ack}_1 \rangle; Y)\ \mathsf{in}\ Y \end{aligned}$$

This implementation imposes exactly the dependencies dictated by the protocol. For example, Alice can receive Bob's acknowledgment to the first 0 before or after Bob receives the first 1. This network extracts to the choreography

$$\mathsf{a}.0 \to \mathsf{b}; X \qquad \text{where} \qquad X = \begin{pmatrix} \mathsf{a}.1 \to \mathsf{b} \\ \mathsf{b}.\mathsf{ack}_0 \to \mathsf{a} \end{pmatrix}; \begin{pmatrix} \mathsf{a}.0 \to \mathsf{b} \\ \mathsf{b}.\mathsf{ack}_1 \to \mathsf{a} \end{pmatrix}; X$$

which is a simple and elegant representation of the alternating 2-bit protocol. □

Extraction for asynchronous SP is still sound, but behavioral equivalence is now an expansion [1,24], as each communication now takes two steps in asynchronous SP. Its complexity is also no larger than for the synchronous case. The algorithm computing the multicom takes linear time in the size of the multicom produced. Via a one-time preprocessing of the network, we can assume direct references from communication terms in one process to the process it directs its communication at, and from there to the current state of that process. Other

than the above, all constant steps in the algorithm can be seen as an extension of the multicom. Since adding a communication to a multicom removes a potential node in the AES (as we are combining communications), the worst-case time complexity is no worse than in the synchronous case. In practice, this complexity actually gets better when larger multicoms are created, since building these is a much cheaper local operation than exploring graphs that would be larger in terms of nodes as well as edges without the multicoms.

6 Extensions and Applications

We discuss some straightforward modifications of our extraction to cover other scenarios occurring in the literature.

More expressive communications and processes. In real-world contexts, the values stored and communicated by processes are typed, and the receiver process can also specify how to treat incoming messages [12]. This means that communication actions now have the form $\mathsf{p}.e \texttt{ -> } \mathsf{q}.f$, where f is the function consuming the received message, and systems may deadlock because of typing errors. Our construction applies without changes to this scenario.

Some works allow processes to store several values, used via variables [5,6]. Again, dealing with this situation does not require any changes to our algorithm.

Local conditionals. Many choreography models allow for a local conditional construct, i.e., $\mathsf{if}\ \mathsf{p}.e\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2$ [6,14,21]. Dealing with this construct is simple: the if and then transitions now can occur whenever a process has a conditional as top action, since they no longer require synchronization with other processes.

Choreography Specifications. So far, we have considered choreographies that describe concrete implementations, i.e., processes are equipped with storage and local computational capabilities. However, choreographies have also been advocated for the specification of communication protocols. Most notably, multiparty session types use choreographies to define types used in the verification of process calculi [17]. While there are multiple variants of multiparty session types, the one so far most used in practice is almost identical to a simplification of SP. In this variant, each pair of participants has a dedicated channel, and communication actions refer directly to the intended sender/recipient as in SP (see, e.g., the theory of [6,9,10,21] and the practical implementations in [16,20,23]). To obtain multiparty session types from SP (and CC), we just need to: remove the capability of storing values at processes; replace message values with constants (representing types, which could also be extended to subtyping in the straightforward way); and make conditionals nondeterministic (since in types we abstract from the precise values and expression used by the evaluator). These modifications do not require any significant change to our approach, since our AES already abstracts from data and thus our treatment of the conditional is already nondeterministic. For reference, we can simply treat the standard construct for an internal choice at a process $\mathsf{p} - C_1 \oplus_{\mathsf{p}} C_2$ – as syntactic sugar for a local conditional like $\mathsf{if}\ \mathsf{p.coinflip}\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2$.

References

1. Arun-Kumar, S., Hennessy, M.: An efficiency preorder for processes. Acta Inf. **29**(8), 737–760 (1992)
2. Basu, S., Bultan, T.: Choreography conformance via synchronizability. In: WWW, pp. 795–804 (2011)
3. Business Process Model and Notation. http://www.omg.org/spec/BPMN/2.0/
4. Brand, D., Zafiropulo, P.: On communicating finite-state machines. J. ACM **30**(2), 323–342 (1983)
5. Carbone, M., Honda, K., Yoshida, N.: Structured communication-centered programming for web services. ACM Trans. Program. Lang. Syst. **34**(2), 8 (2012)
6. Carbone, M., Montesi, F.: Deadlock-freedom-by-design: multiparty asynchronous global programming. In: Giacobazzi, R., Cousot, R. (eds.) POPL, pp. 263–274. ACM (2013)
7. Carbone, M., Montesi, F., Schürmann, C.: Choreographies, logically. In: Baldan, P., Gorla, D. (eds.) CONCUR 2014. LNCS, vol. 8704, pp. 47–62. Springer, Heidelberg (2014). doi:10.1007/978-3-662-44584-6_5
8. Carbone, M., Dardha, O., Montesi, F.: Progress as compositional lock-freedom. In: Kühn, E., Pugliese, R. (eds.) COORDINATION 2014. LNCS, vol. 8459, pp. 49–64. Springer, Heidelberg (2014). doi:10.1007/978-3-662-43376-8_4
9. Carbone, M., Lindley, S., Montesi, F., Schürmann, C., Wadler, P.: Coherence generalises duality: A logical explanation of multiparty session types. In: Desharnais, J., Jagadeesan, R. (eds.) 27th International Conference on Concurrency Theory, CONCUR 2016, August 23–26, 2016, Québec City, Canada, vol. 59 of LIPIcs, pp. 33:1–33:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2016)
10. Coppo, M., Dezani-Ciancaglini, M., Yoshida, N., Padovani, L.: Global progress for dynamically interleaved multiparty sessions. Math. Struct. Comput. Sci. **26**(2), 238–302 (2016)
11. Cruz-Filipe, L., Montesi, F.: A core model for choreographic programming. In: FACS 2016. LNCS. Springer (accepted for publication)
12. Cruz-Filipe, L., Montesi, F.: Choreographies, divided and conquered. CoRR, abs/1602.03729 (2016). Submitted for publication
13. Cruz-Filipe, L., Montesi, F.: Choreographies in practice. In: Albert, E., Lanese, I. (eds.) FORTE 2016. LNCS, vol. 9688, pp. 114–123. Springer, Cham (2016). doi:10.1007/978-3-319-39570-8_8
14. Preda, M., Gabbrielli, M., Giallorenzo, S., Lanese, I., Mauro, J.: Dynamic choreographies. In: Holvoet, T., Viroli, M. (eds.) COORDINATION 2015. LNCS, vol. 9037, pp. 67–82. Springer, Cham (2015). doi:10.1007/978-3-319-19282-6_5
15. Deniélou, P.-M., Yoshida, N.: Multiparty session types meet communicating automata. In: Seidl, H. (ed.) ESOP 2012. LNCS, vol. 7211, pp. 194–213. Springer, Heidelberg (2012). doi:10.1007/978-3-642-28869-2_10
16. Honda, K., Mukhamedov, A., Brown, G., Chen, T.-C., Yoshida, N.: Scribbling interactions with a formal foundation. In: Natarajan, R., Ojo, A. (eds.) ICDCIT 2011. LNCS, vol. 6536, pp. 55–75. Springer, Heidelberg (2011). doi:10.1007/978-3-642-19056-8_4
17. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. J. ACM **63**(1), 9 (2016)
18. Lange, J., Tuosto, E.: Synthesising choreographies from local session types. In: Koutny, M., Ulidowski, I. (eds.) CONCUR 2012. LNCS, vol. 7454, pp. 225–239. Springer, Heidelberg (2012). doi:10.1007/978-3-642-32940-1_17

19. Lange, J., Tuosto, E., Yoshida, N.: From communicating machines to graphical choreographies. In: Rajamani, S.K., Walker, D. (eds.) Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, pp. 221–232. ACM, 15–17 January 2015
20. Montesi, F.: Choreographic Programming. Ph.D. thesis, IT University of Copenhagen (2013). http://fabriziomontesi.com/files/choreographic_programming.pdf
21. Montesi, F., Yoshida, N.: Compositional choreographies. In: D'Argenio, P.R., Melgratti, H. (eds.) CONCUR 2013. LNCS, vol. 8052, pp. 425–439. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40184-8_30
22. Mostrous, D., Yoshida, N., Honda, K.: Global principal typing in partially commutative asynchronous sessions. In: Castagna, G. (ed.) ESOP 2009. LNCS, vol. 5502, pp. 316–332. Springer, Heidelberg (2009). doi:10.1007/978-3-642-00590-9_23
23. Ng, N., Yoshida, N.: Pabble: Parameterised scribble for parallel programming. In: 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2014, Torino, Italy, pp. 707–714. IEEE Computer Society, 12–14 February 2014
24. Sangiorgi, D., Walker, D.: The π-calculus: A Theory of Mobile Processes. Cambridge University Press, Cambridge (2001)
25. W3C WS-CDL Working Group. Web services choreography description language version 1.0 (2004). http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/

On the Undecidability of Asynchronous Session Subtyping

Julien Lange(✉) and Nobuko Yoshida

Imperial College London, London, UK
j.lange@imperial.ac.uk

Abstract. Asynchronous session subtyping has been studied extensively in [9,10,28–31] and applied in [23,32,33,35]. An open question was whether this subtyping relation is decidable. This paper settles the question in the negative. To prove this result, we first introduce a new sub-class of two-party communicating finite-state machines (CFSMs), called asynchronous duplex (ADs), which we show to be Turing complete. Secondly, we give a compatibility relation over CFSMs, which is sound and complete wrt. safety for ADs, and is equivalent to the asynchronous subtyping. Then we show that the halting problem reduces to checking whether two CFSMs are in the relation. In addition, we show the compatibility relation to be decidable for three sub-classes of ADs.

1 Introduction

Session types [22,24,34] specify the expected interaction patterns of concurrent systems and can be used to automatically determine whether communicating processes interact correctly with other processes. A crucial theory in session types is *subtyping* which makes the typing discipline more flexible and therefore easier to integrate in real programming languages and systems [1]. The first subtyping relations for session types targeted synchronous communications [6,7,18,19], by allowing subtypes to make fewer selections and offer more branches. More recent relations treat asynchronous (buffered) communications [9,10,12,13,16,28–31]. They include synchronous subtyping and additionally allow an optimisation by message permutations where outputs can be performed in advance without affecting correctness with respect to the delayed inputs (there are two buffers per session). Only the relative order of outputs (resp. inputs) needs to be preserved to avoid communication mismatches. The asynchronous subtyping is important in parallel and distributed session-based implementations [23,32,33,35], as it reduces message synchronisations without safety violation.

Theoretically, the asynchronous subtyping has been shown to be *precise*, in the sense that: (i) if T is a subtype of U, then a process of type T may be used whenever a process of type U is required and (ii) if T is *not* a subtype of U, then there is a system, requiring a process of type U, for which using a process of type T leads to an error (e.g., deadlock). The subtyping is also denotationally

See [27] for a full version of this paper (with proofs and additional examples).

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 441–457, 2017.
DOI: 10.1007/978-3-662-54458-7_26

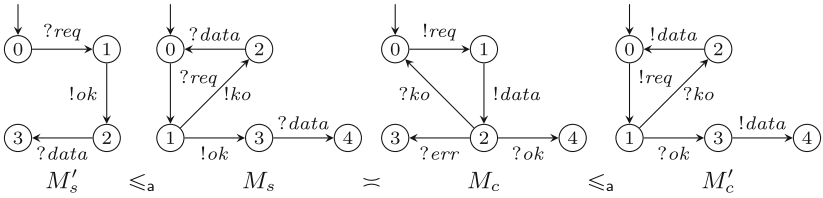


Fig. 1. Asynchronous subtyping and compatibility: examples.

precise taking the standard interpretation of type T as the set of processes typed by T [9,16].

An open question in [9,10,28–31] was whether the asynchronous subtyping relation is decidable, i.e., is there an algorithm to decide whether two types are in the relation. The answer to that question was thought to be positive, see [10, Sect. 7] and Sect. 6.

Asynchronous Subtyping, Informally. In this work, we consider session types in the form of CFSMs [4], along the lines of [3,14,15,25]. This enables us to characterise the asynchronous subtyping in CFSMs and reduce the undecidability problem to the Turing completeness of CFSMs. Consider a system of CFSMs consisting of machines M_s (server) and M_c (client) in Fig. 1, which communicate via two unbounded queues, one in each direction. A transition $!a$ represents the (asynchronous) emission of a message a, while $?a$ represents the receptions of a message a from a buffer. For instance, the transition labelled by $!req$ in M_c says that the client sends a request to the server M_s, later the server can consume this message from its buffer by firing the transition labelled by $?req$. We say that the system (M_s, M_c), i.e., the parallel composition of M_s and M_c, is *safe* if (i) the pair never reaches a deadlock and (ii) whenever a message is sent by one party, it will eventually be received by the other.

The key property of session subtyping is that, e.g., if the system (M_s, M_c') is safe and M_c is a subtype of M_c', the system (M_s, M_c) is also safe. We write $\leqslant_a$ for the asynchronous subtyping relation, which intuitively requires that, if, e.g., $M_c \leqslant_a M_c'$, then M_c is ready to receive no fewer messages than M_c' and it may not send more messages than M_c'. For instance, M_c can receive all the messages that M_c' can handle, plus the message err. Observe that M_c is an optimised version of M_c' wrt. asynchrony: the output action $!data$ is performed in advance of the branching. Thus in the system (M_s, M_c), when both machines are in state 2 (respectively), both queues contain messages. Instead, in the system (M_s, M_c'), it is never the case that both queues are non-empty. Note that anticipating the sending of $data$ in M_c does not affect safety as it is sent in both branches of M_c'.

Our Approach. Using CFSMs, we give the first automata characterisation of asynchronous subtyping and the first proof of its undecidability. To do this, we introduce a new sub-class of CFSMs, called *asynchronous duplex* (AD) which let

us study directly the relationship between safety and asynchronous subtyping in CFSMs. Our development consists of the following steps:

Step 1. In Sect. 2, we define a new sub-class of (two-party) CFSMs, called asynchronous duplex (AD), which strictly includes *half-duplex* (HD) systems [8].
Step 2. In Sect. 3, we introduce a compatibility relation ($\asymp$) for CFSMs which is sound and complete wrt. safety in AD CFSMs, i.e., an AD system has no deadlocks nor orphan messages if and only if its machines are $\asymp$-related.
Step 3. Adapting the result of [17], we show in Sect. 4 that AD systems are Turing complete, hence membership of $\asymp$ is generally undecidable.
Step 4. In Sect. 5, we show that the $\asymp$-relation for CFSMs is equivalent to the asynchronous subtyping for session types, thus establishing that the latter is also undecidable.

Throughout the paper, we also show that our approach naturally encompasses the correspondence between synchronous subtyping and safety in HD systems.

In Sect. 4.1, we show that the $\asymp$-relation is decidable for three sub-classes of CFSMs (HD, alternating [21], and non-branching) which are useful to specify real-world protocols. In Sect. 6, we discuss related works and conclude.

2 A New Class of CFSMs: Asynchronous Duplex Systems

This section develops **Step 1** by defining a new sub-class of CFSMs, called *asynchronous duplex*, which characterises machines that can only simultaneously write on their respective channels if they can only do so for finitely many consecutive send actions before executing a receive action. In Sect. 2.1, we recall definitions about CFSMs, then we give the definition of safety. In Sect. 2.2, we introduce the sub-class of AD systems and give a few examples of such systems.

2.1 CFSMs and Their Properties

Let $\mathbb{A}$ be a (finite) alphabet, ranged over by a, b, etc. We let ω, π, and φ range over words in $\mathbb{A}^*$ and write $\cdot$ for the concatenation operator. The set of actions is $Act = \{!, ?\} \times \mathbb{A}$, ranged over by ℓ, $!a$ represents the emission of a message a, while $?a$ represents the reception of a. We let ψ range over Act^* and define $dir(!a) \stackrel{\text{def}}{=} !$ and $dir(?a) \stackrel{\text{def}}{=} ?$.

Since our ultimate goal is to relate CFSMs and session types, we only consider deterministic communicating finite-state machines, without mixed states (i.e., states that can fire both send and receive actions) as in [14,15].

Definition 2.1 (Communicating machine). *A (communicating) machine M is a tuple (Q, q_0, δ) where Q is the (finite) set of states, $q_0 \in Q$ is the initial state, and $\delta \subseteq Q \times Act \times Q$ is the transition relation such that $\forall q, q', q'' \in Q : \forall \ell, \ell' \in Act :$ (1) $(q, \ell, q'), (q, \ell', q'') \in \delta \implies dir(\ell) = dir(\ell')$, and (2) $(q, \ell, q'), (q, \ell, q'') \in \delta \implies q' = q''$.*

We write $q \xrightarrow{\ell} q'$ for $(q, \ell, q') \in \delta$, omit the label ℓ when unnecessary, and write $\rightarrow^*$ for the reflexive transitive closure of $\rightarrow$.

Given $M = (Q, q_0, \delta)$, we say that $q \in Q$ is *final*, written $q \not\to$, iff $\forall q' \in Q : \forall \ell \in Act : (q, \ell, q') \notin \delta$. A state $q \in Q$ is *sending* (resp. *receiving*) iff q is not final and $\forall q' \in Q : \forall \ell \in Act : (q, \ell, q') \in \delta : dir(\ell) = !$ (resp. $dir(\ell) = ?$). The dual of M, written $\overline{M}$, is M where each sending transition $(q, !a, q') \in \delta$ is replaced by $(q, ?a, q')$, and vice-versa for receive transitions, e.g., $\overline{M}_s = M'_c$ in Fig. 1.

We write $q_0 \xrightarrow{\ell_1 \cdots \ell_k} q_k$ iff there are $q_1, \ldots, q_{k-1} \in Q$ such that $q_{i-1} \xrightarrow{\ell_i} q_i$ for $1 \leq i \leq k$. Given a list of messages $\omega = a_1 \cdots a_k$ ($k \geq 0$), we write $?\omega$ for the list $?a_1 \cdots ?a_k$ and $!\omega$ for $!a_1 \cdots !a_k$. We write $q \xrightarrow{!}{}^* q'$ iff $\exists \omega \in \mathbb{A}^* : q \xrightarrow{!\omega} q'$ and $q \xrightarrow{?}{}^* q'$ iff $\exists \omega \in \mathbb{A}^* : q \xrightarrow{?\omega} q'$ (note that ω may be empty, in which case $q = q'$).

Definition 2.2 (System). *A system $S = (M_1, M_2)$ is a pair of machines $M_i = (Q_i, q_{0_i}, \delta_i)$ with $i \in \{1, 2\}$.*

Hereafter, we fix $S = (M_1, M_2)$ and assume $M_i = (Q_i, q_{0_i}, \delta_i)$ for $i \in \{1, 2\}$ such that $Q_1 \cap Q_2 = \varnothing$. Hence, for $q, q' \in Q_i$, we can write $q \xrightarrow{\ell} q'$ to refer unambiguously to δ_i.

We let λ range over the set $\{{}_{ij}!a \mid i \neq j \in \{1, 2\}\} \cup \{{}_{ij}?a \mid i \neq j \in \{1, 2\}\}$ and ϕ range over (possibly empty) sequences of $\lambda_1 \cdots \lambda_k$.

Definition 2.3 (Reachable configuration). *A configuration of S is a tuple $s = (q_1, \omega_1, q_2, \omega_2)$ such that $q_i \in Q_i$, and $\omega_i \in \mathbb{A}^*$. A configuration $s' = (q'_1, \omega'_1, q'_2, \omega'_2)$ is reachable from $s = (q_1, \omega_1, q_2, \omega_2)$, written $s \overset{\lambda}{\Rightarrow} s'$, iff*

1. *$q_i \xrightarrow{!a} q'_i$, $\omega'_i = \omega_i \cdot a$, $q_j = q'_j$, and $\omega_j = \omega'_j$, $\lambda = {}_{ij}!a$, for $i \neq j \in \{1, 2\}$, or*
2. *$q_i \xrightarrow{?a} q'_i$, $\omega_j = a \cdot \omega'_j$, $q_j = q'_j$, and $\omega_i = \omega'_i$, $\lambda = {}_{ji}?a$, for $i \neq j \in \{1, 2\}$.*

We write $s \Rightarrow s'$ when the label is irrelevant and $\Rightarrow^*$ for the reflexive and transitive closure of $\Rightarrow$.

In Definition 2.3, (1) says that machine M_i puts a message on queue i, to be received by machine M_j, while (2) says that machine M_i consumes a message from queue j, which was sent by M_j.

Given a system S, we write s_0 for its initial configuration $(q_{0_1}, \epsilon, q_{0_2}, \epsilon)$ and let $RS(S) \overset{\text{def}}{=} \{s \mid s_0 \Rightarrow^* s\}$.

Definition 2.4 (Safety). *A configuration $s = (q_1, \omega_1, q_2, \omega_2)$ is a* deadlock *iff $\omega_1 = \omega_2 = \epsilon$, q_i is a receiving state, and q_j is either receiving or final for $i \neq j \in \{1, 2\}$. System S satisfies* eventual reception *iff $\forall s = (q_1, \omega_1, q_2, \omega_2) \in RS(S) : \forall i \neq j \in \{1, 2\} : \omega_i \in a \cdot \mathbb{A}^* \implies \forall q'_j \in Q_j : q_j \xrightarrow{!}{}^* q'_j \implies q'_j \xrightarrow{!}{}^* \xrightarrow{?a}$.*

S is safe *iff (i) for all $s \in RS(S)$, s is not a deadlock, and (ii) S satisfies eventual reception (i.e., every sent message is eventually received).*

Lemma 2.1 below shows that safety implies progress and that a configuration with at least one empty buffer is always reachable.

Lemma 2.1. *If S is* safe, *then for all $s = (q_1, \omega_1, q_2, \omega_2) \in RS(S)$*

1. *Either (i) q_1 and q_2 are final and $\omega_1 = \omega_2 = \epsilon$, or (ii) $\exists s' \in RS(S) : s \Rightarrow s'$.*
2. *$\exists s', s'' \in RS(S) : s \Rightarrow^* s' = (q_1, \epsilon, q'_2, \omega_2 \cdot \omega'_2) \wedge s \Rightarrow^* s'' = (q''_1, \omega_1 \cdot \omega''_1, q_2, \epsilon)$.*

Fig. 2. Examples of AD (left) and non-AD (right) systems.

2.2 Asynchronous Duplex Systems

We define asynchronous duplex systems, a sub-class of two-party CFSMs. Below we introduce a predicate which guarantees that when a machine is in a given state, it cannot send infinitely many messages without executing receive actions periodically. This predicate mirrors one of the premises of the defining rules of the asynchronous subtyping ($\leqslant_{\mathtt{a}}$), cf. Lemma 5.1. Given $M = (Q, q_0, \delta)$ and $q \in Q$, we define $\mathtt{fin}(q) \iff \mathtt{fin}(q, \varnothing)$, where

$$\mathtt{fin}(q, R) \stackrel{\text{def}}{=} \begin{cases} \mathit{true} & \text{if } q \xrightarrow{?a} \\ \forall q' \in \{q' \mid q \xrightarrow{!a} q'\} : \mathtt{fin}(q', R \cup \{q\}) & \text{if } q \xrightarrow{!a} \wedge\, q \notin R \\ \mathit{false} & \text{otherwise} \end{cases}$$

Definition 2.5 (Asynchronous duplex). *A system $S = (M_1, M_2)$ is* Asynchronous Duplex *(AD) if for each $s = (q_1, \omega_1, q_2, \omega_2) \in RS(S) : \omega_1 \neq \epsilon \wedge \omega_2 \neq \epsilon \implies \mathtt{fin}(q_1) \wedge \mathtt{fin}(q_2)$.*

AD systems are a strict extension of half-duplex systems [8]: S is half-duplex (HD) if for all $(q_1, \omega_1, q_2, \omega_2) \in RS(S) : \omega_1 = \epsilon \vee \omega_2 = \epsilon$. AD requires that for any reachable configuration either (i) at most one channel is non-empty (i.e., it is a half-duplex configuration) or (ii) each machine is in a state where the predicate $\mathtt{fin}(\_)$ holds, i.e., each machine will reach a receiving state after firing *finitely* many send actions. The AD restriction is reasonable for real-word systems. It intuitively amounts to say that if two parties are *simultaneously* sending data to each other, they should both ensure that they will periodically check what the other party has been sending.

Example 2.1. Consider the machines in Fig. 2. The system (M_1, M_2) is AD: $\mathtt{fin}(\_)$ holds for each state in M_1 and M_2. The system $(\hat{M}_1, \hat{M}_2)$ is not AD. For instance, the configuration $(0, a, 0, c)$ is reachable but we have $\neg\mathtt{fin}(0)$ for both initial states of $\hat{M}_1$ and $\hat{M}_2$. Observe that both systems are *safe*, cf. Definition 2.4.

3 A Compatibility Relation for CFSMs

This section develops **Step 2**: we introduce a binary relation $\asymp$ on CFSMs which is sound and complete wrt. safety (cf. Definition 2.4) for AD systems. That is $M_1 \asymp M_2$ holds if and only if (M_1, M_2) is a safe asynchronous duplex system.

Definition 3.1 (Compatibility). *Let* $M_i = (Q_i, q_{0_i}, \delta_i)$ *for* $i \in \{1,2\}$ *such that* $Q_1 \cap Q_2 = \varnothing$, *and let* $p \in Q_1$, $q \in Q_2$, *and* $\pi \in \mathbb{A}^*$.

The compatibility *relation is defined as follows:* $\pi \blacktriangleright p \asymp_0 q$ *always holds, and if* $k \geq 0$, *then* $\pi \blacktriangleright p \asymp_{k+1} q$ *holds iff*

1. *if* $p \not\to$ *then* $\pi = \epsilon$ *and* $q \not\to$
2. *if* $p \xrightarrow{?a}$ *then*
   (a) *if* $\pi = \epsilon$ *then,* $q \xrightarrow{!b}$ *and* $\forall b \in \mathbb{A} : q \xrightarrow{!b} q' \implies (p \xrightarrow{?b} p' \wedge \epsilon \blacktriangleright p' \asymp_k q')$,
   (b) *if* $\pi = b \cdot \pi'$ *then,* $\exists p' \in Q_1 : p \xrightarrow{?b} p' \wedge \pi' \blacktriangleright p' \asymp_k q$
3. *if* $p \xrightarrow{!a} p'$ *then either*
   (a) $\pi = \epsilon$ *and* $\exists q' \in Q_2 : q \xrightarrow{?a} q' \wedge \epsilon \blacktriangleright p' \asymp_k q'$, *or*
   (b) $\mathtt{fin}(p)$, $\mathtt{fin}(q)$, *and* $\forall q' \in Q_2 : \forall \pi' \in \mathbb{A}^* : q \xrightarrow{!\pi'} q'$, *there exist* $\pi'' \in \mathbb{A}^*$ *and* $q'' \in Q_2$ *such that* $q' \xrightarrow{!\pi''} \xrightarrow{?a} q''$ *and* $\pi \cdot \pi' \cdot \pi'' \blacktriangleright p' \asymp_k q''$

Define $\pi \blacktriangleright p \asymp q \stackrel{def}{=} \forall k \in \mathbb{N} : \pi \blacktriangleright p \asymp_k q$ *and* $M_1 \asymp M_2 \stackrel{def}{=} \epsilon \blacktriangleright q_{0_1} \asymp q_{0_2}$.

The relation $M_1 \asymp M_2$ checks that the two machines are compatible by executing M_1 while recording what M_2 asynchronously sends to M_1 in the π message list. The definition first differentiates the type of state p:

Final. Case (1) says that if M_1 is in a final state, then M_2 must also be in a final state and π must be empty (i.e., M_1 has emptied its input buffer).

Receiving. Case (2) says that if M_1 is in a receiving state, then either π is empty and M_1 must be ready to receive any message sent by M_2, cf. case (2a); otherwise, case (2b) must apply: M_1 must consume the head of the message list π, this models the FIFO consumption of messages sent by M_2.

Sending. Case (3) says that if M_1 is ready to send a, then either M_2 must be able to receive a directly, cf. case (3a). Otherwise, $\mathtt{fin}(p)$ and $\mathtt{fin}(q)$ must hold so that case (3b) applies. M_2 may delay the reception of a by sending messages (which are recorded in $\pi' \cdot \pi''$). Whichever sending path M_2 chooses, it must always eventually receive a.

We write $\asymp_s$ for the *synchronous compatibility relation*, i.e., Definition 3.1 without cases (2b) and (3b).

Example 3.1. **(1)** Recall the machines from Fig. 1, we have $M_s \asymp M_c$, in particular: $\epsilon \blacktriangleright 0 \asymp 0$ and $data \blacktriangleright 2 \asymp 0$. The latter relation represents the fact that M_c and M_s have exchanged the messages *req* and *ko*, but M_s has yet to process the reception of *data*. Observe that we also have $M_s' \asymp M_c'$ and $M_s' \asymp_s M_c'$.
(2) Consider the systems in Fig. 2. We have $M_1 \asymp M_2$ and $\hat{M}_1 \not\asymp \hat{M}_2$. The latter does not hold since both initial states are sending states, but the predicate $\mathtt{fin}(\_)$ does not hold for either state, e.g., we have $\neg\mathtt{fin}(0, \{0\})$ in $\hat{M}_1$.

Soundness of $\preceq$. We show the soundness of the $\preceq$-relation wrt. safety. More precisely we show that if $M_1 \preceq M_2$ holds, then the system (M_1, M_2) is a safe AD system. We first give two auxiliary definitions which are convenient to relate safety with the definition of $\preceq$. Fixing $M = (Q, q_0, \delta)$, the predicate $A(q, \omega)$ asserts when a list of messages ω is "accepted" from a state $q \in Q$, which implies eventual reception of the messages in ω. The function $W(q, \omega)$ is used to connect a configuration to a triple in the $\preceq$-relation.

Definition 3.2. *Let $q \in Q$ and $\omega \in \mathbb{A}^*$, we define*

$$A(q,\omega) \iff \begin{cases} \forall q' : q \xrightarrow{!}{}^* q' : \exists \hat{q} : q' \xrightarrow{!}{}^* \xrightarrow{?a} \hat{q} \wedge A(\hat{q}, \omega') & if\ \omega = a \cdot \omega' \\ true & if\ \omega = \epsilon \end{cases}$$

Given $q \in Q$ and $\omega \in \mathbb{A}^*$, the predicate $A(q, \omega)$ is true iff the list of messages ω can always be consumed entirely from state q, for all paths reachable from q by send actions. Note the similarity with case (3b) of Definition 3.1.

Definition 3.3. *Let $q \in Q$ and $\omega \in \mathbb{A}^*$, $W(q, \omega) \subseteq \mathbb{A}^* \times Q$ is the set such that*

$$(\pi, \hat{q}) \in W(q,\omega) \iff \begin{cases} (\varphi, \hat{q}) \in W(q', \omega') & if\ \omega = a \cdot \omega', q \xrightarrow{!\pi' \cdot ?a} q', \pi = \pi' \cdot \varphi \\ \pi = \epsilon \wedge \hat{q} = q & if\ \omega = \epsilon \end{cases}$$

Each pair $(\pi, \hat{q})$ in $W(q, \omega)$ represents a state $\hat{q} \in Q$ reachable directly after having consumed the list of messages ω, while π is the list of messages that are *sent* along a path between q and $\hat{q}$. For example, consider M_c from Fig. 1. We have $A(0, ko \cdot ko \cdot err)$ and $W(0, ko \cdot ko \cdot err) = \{(req \cdot data \cdot req \cdot data, 3)\}$; instead, $\neg A(0, ok \cdot ko)$ and $\neg A(4, ko)$.

Lemma 3.1. *Let $M = (Q, q_0, \delta)$, $q \in Q$ and $\omega \in \mathbb{A}^*$. If $A(q, \omega)$ and $\forall (\varphi, q') \in W(q, \omega) : A(q', a)$ then $A(q, \omega \cdot a)$.*

Lemma 3.1, shown by induction on the size of ω, is useful in the proof of the main soundness lemma below.

Lemma 3.2. *Let $S = (M_1, M_2)$. If $M_1 \preceq M_2$, then for all $s = (p, \omega_1, q, \omega_2) \in RS(S)$ the following holds:* (1) *s is not a deadlock,* (2) $A(q, \omega_1)$, (3) $\forall (\varphi, q') \in W(q, \omega_1) : \omega_2 \cdot \varphi \blacktriangleright p \preceq q'$, *and* (4) $A(p, \omega_2)$.

Lemma 3.2 states that for any configuration s: (1) s is not a deadlock; (2) M_2 can consume the list ω_1 from state q; (3) for each state q', reached after consuming ω_1, the relation $\omega_2 \cdot \varphi \blacktriangleright p \preceq q'$ holds, where φ contains the messages that M_2 sent while consuming ω_1; and (4) M_1 can consume the list ω_2 from state p. The proof of Lemma 3.2 is by induction on the length of an execution from s_0 to s, then by case analysis on the last action fired to reach s. Lemma 3.1 is used for the case $s_0 \Rightarrow^* \xrightarrow{12!a} s$, i.e., to show that $A(q, \omega_1 \cdot a)$ holds.

Lemma 3.3. *Let $S = (M_1, M_2)$. If for all $s = (q_1, \omega_1, q_2, \omega_2) \in RS(S)$: $A(q_1, \omega_1)$ and $A(q_2, \omega_2)$, then S satisfies* eventual reception.

Lemma 3.3 simply shows a correspondence between eventual reception and Definition 3.2. The proof essentially shows that if $A(q_i, \omega_j)$ holds, then we can always reach a configuration where the list ω_j has been entirely consumed.

Finally, we state our final soundness results. Theorem 3.1 is a consequence of Lemmas 2.1, 3.2, 3.3, and 3.4. Theorem 3.2 essentially follows from Theorem 3.1 and the fact that $\asymp_s \subseteq \asymp$.

Theorem 3.1. *If $M_1 \asymp M_2$, then (M_1, M_2) is a* safe AD *system.*

Theorem 3.2. *If $M_1 \asymp_s M_2$, then (M_1, M_2) is a* safe HD *system.*

Completeness of $\asymp$. Our completeness result shows that for any safe asynchronous duplex system $S = (M_1, M_2)$, $M_1 \asymp M_2$ holds. Below we show that any reachable configuration of S whose first queue is empty can be mapped to a triple that is in the relation of Definition 3.1.

Lemma 3.4. *Let S be* safe *and* AD*, then $\forall (p, \epsilon, q, \omega) \in RS(S) : \omega \blacktriangleright p \asymp q$.*

The proof of Lemma 3.4 is by induction on the k^{th} approximation of $\asymp$, i.e., assuming that $\omega \blacktriangleright p \asymp_k q$ holds, we show that $\omega \blacktriangleright p \asymp_{k+1} q$ holds. The proof is a rather straightforward case analysis on the type of p and whether or not $\omega = \epsilon$.

Theorem 3.3. *If (M_1, M_2) is a* safe AD *system, then $M_1 \asymp M_2$.*

Proof. Take $(q_{0_1}, \epsilon, q_{0_2}, \epsilon) \in RS(S)$, $\epsilon \blacktriangleright q_{0_1} \asymp q_{0_2}$ holds by Lemma 3.4. □

Following a similar (but simpler) argument, we have Theorem 3.4 below.

Theorem 3.4. *If (M_1, M_2) is a* safe HD *system, then $M_1 \asymp_s M_2$.*

Theorem 3.5. *If $M_1 \asymp M_2$ (resp. $M_1 \asymp_s M_2$), then $M_2 \asymp M_1$ (resp. $M_2 \asymp_s M_1$).*

Proof. We show the $\asymp$ part. By Theorem 3.1, (M_1, M_2) is safe, hence by definition of safety, (M_2, M_1) is also safe. Thus by Theorem 3.3, we have $M_2 \asymp M_1$. □

4 Undecidability of the $\asymp$-relation

This section addresses **Step 3**: we show that the problem of checking $M_1 \asymp M_2$ is undecidable. We show that AD systems are Turing complete, then show that the halting problem reduces to deciding whether or not a system is safe.

Preliminaries. We adapt the relevant part of the proof of Finkel and McKenzie [17] to demonstrate that the problem of deciding whether two machines are $\asymp$-related is undecidable. For this we need to show that there is indeed a Turing machine encoding that is an AD system.

Definition 4.1 (Turing machine [17]). *A Turing machine (T.M.) is a tuple $TM = (V, \mathbb{A}, \Gamma, t_0, \mathtt{B}, \gamma)$ where V is the set of states, $\mathbb{A}$ is the input alphabet, Γ is the tape alphabet, $t_0 \in V$ is the initial state, $\mathtt{B}$ is the blank symbol, and $\gamma : V \times \Gamma \to V \times \Gamma \times \{left, right\}$ is the (partial) transition function.*

Assume TM accepts an input $\omega \in \mathbb{A}^*$ iff TM halts on ω, and if TM does not halt on ω, then TM eventually moves its tape head arbitrarily far to the right.

Definition 4.2 (Configuration of a T.M. [17]). *A configuration of the Turing machine TM is a word $\omega_1 t \omega_2 \#$ with $\omega_1\omega_2 \in \mathbb{A}^*$, $t \in V$, and $\# \notin \Gamma$.*

The word $\omega_1 t \omega_2 \#$ represents TM in state $t \in V$ with the tape content set to $\omega_1\omega_2$ and the rest blank, and TM's head positioned under the first symbol to the right of ω_1. Symbol # is a symbol used to mark the end of the tape.

T.M. Encoding. We present an AD system which encodes a Turing machine $TM = (V, \mathbb{A}, \Gamma, t_0, \mathcal{B}, \gamma)$ with initial tape ω into a system of two CFSMs as in [17].

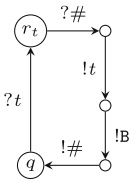
We explain the T.M. encoding. The two channels represent the tape of the Turing machine, with a marker # separating the two ends of the tape. Each machine represents the control of the Turing machine as well as a transmitter from a channel to another. The head is represented by writing the current control state $t \in V$ on the channel. Whenever a machine receives a message that is $t \in V$, then it proceeds with one execution step of the Turing machine. Any other symbol is simply consumed from one channel and sent on the other. The only difference wrt. [17] is that we construct machines which are deterministic and which do not contain mixed states, cf. Definition 2.1. We also do not require the machines to be identical hence we encode the initial tape content as a sequence of transitions in the first machine. These slight modifications do not affect the rest of Finkel and McKenzie's proof in [17]. The system consists of two CFSMs $A_i = (Q_i, q_{0_i}, \delta_i)$, $i \in \{1, 2\}$ over the alphabet $\mathbb{A} \cup \{\#\}$. The definitions of δ_i are given below, the sets Q_i are induced by δ_i. The transition relation δ_1 consists in a sequence of transitions from the initial state q_{0_1} to a central state q and a number of elementary cycles around state q, cf. Fig. 3; while δ_2 is like δ_1 without the initial sequence of transitions and $q = q_{0_2}$. The initial sequence of transitions in δ_1 is of the form:

$$q_{0_1} \xrightarrow{!t_0} q_1 \xrightarrow{!a_1} \cdots q_k \xrightarrow{!a_k} q \qquad \text{such that } a_1 \cdots a_k = \omega \cdot \#$$

Both δ_1 and δ_2 contain six types of elementary cycles given in Fig. 3. For each type of cycle, we illustrate the behaviour of the system from the point view of machine A_2 by giving the type of configuration this cycle applies to as well the configuration obtained after A_2 has finished executing the cycle.

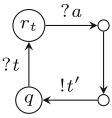
When computing each δ_i and Q_i from the description above, we assume that each "anonymous" state maintain its own identity, while "named" states, i.e., q, r_t, r_x and r_x^t from Fig. 3, are to be identified and redundant transitions to be removed. This ensures that each machine so obtained is deterministic. Besides this determinisation, the only changes from [17] concerns the copying cycles. (1) Each copying cycle is expanded to receive (then send) two symbols so to ensure the absence of mixed states once merged with left head motion cycles. (2) We add a cycle which only re-emits # symbols (to make up for absence of it in the first reception of the copying cycles). (3) We add another blank insertion cycle to deal with the special case where the head is followed by the # symbol.

Blank insertion cycles (1). For each $t \in V$, there is a cycle of the form:



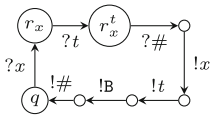
$(q', t \cdot \# \cdot \omega_1, q, \omega_2) \Rightarrow^* (q', \omega_1, q, \omega_2 \cdot t \cdot \mathtt{B} \cdot \#)$

Right head motion cycles. For each $(t, a, t', b) \in V \times \Gamma \times V \times \Gamma$ such that $\gamma(t, a) = (t', b, \mathit{right})$, there is a cycle of the form:



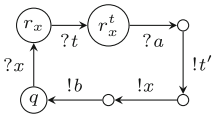
$(q', t \cdot a \cdot \omega_1, q, \omega_2) \Rightarrow^* (q', \omega_1, q, \omega_2 \cdot b \cdot t')$

Blank insertion cycles (2). For each $x \in \Gamma$ and $t \in V$ there is a cycle of the form:



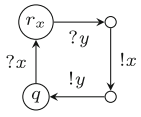
$(q', x \cdot t \cdot \# \cdot \omega_1, q, \omega_2) \Rightarrow^* (q', \omega_1, q, \omega_2 \cdot x \cdot t \cdot \mathtt{B} \cdot \#)$

Left head motion cycles. For each $(x, t, a, t', b) \in \Gamma \times V \times \Gamma \times V \times \Gamma$ such that $\gamma(t, a) = (t', b, \mathit{left})$, there is a cycle of the form:



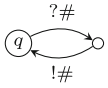
$(q', x \cdot t \cdot a \cdot \omega_1, q, \omega_2) \Rightarrow^* (q', \omega_1, q, \omega_2 \cdot t' \cdot x \cdot b)$

Copying cycles. For all $x \in \Gamma$ and $y \in \Gamma \cup \{\#\}$, there is a cycle of the form:



$(q', x \cdot y \cdot \omega_1, q, \omega_2) \Rightarrow^* (q', \omega_1, q, \omega_2 \cdot x \cdot y)$

Marker transmission cycle. There is one cycle specified by:



$(q', \# \cdot \omega_1, q, \omega_2) \Rightarrow^* (q', \omega_1, q, \omega_2 \cdot \#)$

Fig. 3. Definition of δ_i (elementary cycles).

Definition 4.3 (Turing machine encoding [17]). *Given a Turing machine TM and an initial tape content ω, we write $S(TM, \omega)$ for the system (A_1, A_2) with each A_i constructed as described above.*

The rest follows the proof of [17]. Here we recall informally the final result: any execution of a Turing machine TM with initial word ω can be simulated by $S(TM, \omega)$, and vice-versa.

Lemma 4.1. *For any TM and word ω, $S(TM, \omega) = (A_1, A_2)$ is* AD.

Proof. Take $A_i = (Q_i, q_{0_i}, \delta_i)$, we show $\forall q \in Q_i : \mathtt{fin}(q)$, which implies that the system is AD. If there was $q \in Q_i$ such that $\neg\mathtt{fin}(q)$, there would a cycle of send actions only, the construction of A_i clearly prevents this (see Fig. 3). □

Theorem 4.1 (Undecidability of $\asymp$). *Given two machines M_1 and M_2, it is generally undecidable whether $M_1 \asymp M_2$ holds.*

The proof of Theorem 4.1 shows that the following statements are equivalent: (1) TM accepts ω, (2) $S(TM, \omega) = (A_1, A_2)$ is *not* safe, and (3) $\neg(A_1 \preceq A_2)$. We show (1) $\Rightarrow$ (2) by Lemma 2.1, (2) $\Rightarrow$ (1) from the definition of safety, and (2) $\Leftrightarrow$ (3) by Theorems 3.1 and 3.3 and the fact that (A_1, A_2) is AD.

4.1 Decidable Sub-classes of CFSMs

We now identify three sub-classes of CFSMs for which the $\preceq$-relation is decidable. We say that $M_1 \preceq M_2$ is decidable iff it is decidable whether or not $M_1 \preceq M_2$ holds. The first sub-class is HD systems: HD is a decidable property and safety is decidable within that class [8], hence $\preceq$ is decidable in HD and it is equivalent to $\preceq_s$ within HD. The second sub-class is taken from the CFSMs literature and the third is limited to systems that contain at least one machine that has no branching. We define the last two sub-classes below.

The following definition is convenient to formalise our decidability results. Given $M_i = (Q_i, q_{0_i}, \delta_i)$ for $i \in \{1, 2\}$, the *derivation tree* of a triple $\pi \blacktriangleright p \preceq q$ is a tree whose nodes are labelled by elements of $\mathbb{A}^* \times Q_1 \times Q_2$ such that the children of a node are exactly the triple generated by applying one step of Definition 3.1.

For example, consider the machines M_1 and M_2 from Fig. 2, we have a tree which consists of a unique (infinite) branch:

$$\epsilon \blacktriangleright 0 \preceq 0 \longrightarrow b \blacktriangleright 1 \preceq 0 \longrightarrow bb \blacktriangleright 2 \preceq 0 \longrightarrow b \blacktriangleright 0 \preceq 0 \longrightarrow bb \blacktriangleright 1 \preceq 0 \longrightarrow bbb \blacktriangleright 2 \preceq 0 \cdots$$

Lemma 4.2. *The derivation tree of $\pi \blacktriangleright p \preceq q$ is finitely branching.*

Lemma 4.2 follows from the fact that each machine is finitely branching and the predicate $\texttt{fin}(\_)$ guarantees finiteness for case (3b) of Definition 3.1.

Alternating Machines. Alternating machines were introduced in [21] where it is shown that the progress problem (corresponding to our notion of safety) is decidable for such systems. A machine is *alternating* if each of its sending transition is followed by a receiving transition, e.g., M_s and M'_s in Fig. 1 are alternating, as well as the specification of the alternating-bit protocol in [21]. Observe that alternating machines form AD systems.

Theorem 4.2. *If M_1 and M_2 are* alternating*, then $M_1 \preceq M_2$ is decidable.*

The proof simply shows that the π part of the relation (cf. Definition 3.1) is bounded by 1, by induction on the depth of the derivation tree.

Non-branching Machines. Given $M = (Q, q_0, \delta)$ we say that M is non-branching if each of its state has at most one successor, i.e., if $\forall q \in Q : |\delta(q)| \leq 1$. For example, M'_s in Fig. 1 is non-branching. Non-branching machines are used notably in [33,35] to specify parallel programs which can be optimised through asynchronous message permutations.

Theorem 4.3. *Let M_1 and M_2 be two machines such that at least one of them is* non-branching*, then $M_1 \asymp M_2$ is decidable.*

The proof relies on the fact that (*i*) the derivation tree is finitely branching (Lemma 4.2), hence there is a semi-algorithm to checker whether $\neg(M_1 \asymp M_2)$ and (*ii*) over any infinite branches we can find two nodes of the form $\mathtt{c} = \pi^n \blacktriangleright p \asymp q$ and $\mathtt{c}' = \pi^m \blacktriangleright p \asymp q$, with $n \leq m$. If n is large enough, this implies that the relation holds (i.e., the branch is indeed infinite).

5 Correspondence Between Compatibility and Subtyping

We show a precise correspondence between the asynchronous subtyping for session types and the $\asymp$-relation for CFSMs, i.e., **Step 4**. We first recall the syntax of session types and as well as the definition of asynchronous subtyping.

Session Types and Subtyping. The syntax of session types is given by

$$T, U \coloneqq \quad \mathtt{end} \quad | \quad \oplus_{i\in I} ! a_i . T_i \quad | \quad \&_{i\in I} ? a_i . T_i \quad | \quad \mathtt{rec}\, \mathbf{x}.T \quad | \quad \mathbf{x}$$

where $I \neq \varnothing$ is finite and $a_i \neq a_j$ for $i \neq j$. Type $\mathtt{end}$ indicates the end of a session. Type $\oplus_{i\in I} ! a_i . T_i$ specifies an *internal* choice, indicating that the program chooses to send one of the a_i messages, then behaves as T_i. Type $\&_{i\in I} ? a_i . T_i$ specifies an *external* choice, saying that the program waits to receive one of the a_i messages, then behaves as T_i. Types $\mathtt{rec}\, \mathbf{x}.T$ and $\mathbf{x}$ are used to specify recursive behaviours. We only consider closed types, i.e., without free variables.

Since our goal is to relate a binary relation defined on CFSMs to a binary relation on session types, we first introduce transformations from one to another.

Definition 5.1. *Given a type T, we write $\mathcal{M}(T)$ for the CFSM induced by T. Given a CFSM M, we write $\mathcal{T}(M)$ for the type constructed from M.*

We assume the existence of two algorithms such that $T = \mathcal{T}(\mathcal{M}(T))$ and $M = \mathcal{M}(\mathcal{T}(M))$ for any type T and machine M. These algorithms are rather trivial since each session type induces a finite automaton, see [15] for instance.

We write $\overline{T}$ for the *dual* of type T, i.e., $\overline{\mathtt{end}} = \mathtt{end}$, $\overline{\mathbf{x}} = \mathbf{x}$, $\overline{\mathtt{rec}\, \mathbf{x}.T} = \mathtt{rec}\, \mathbf{x}.\overline{T}$, $\overline{\oplus_{i\in I} ! a_i . T_i} = \&_{i\in I} ? a_i . \overline{T}_i$, and $\overline{\&_{i\in I} ? a_i . T_i} = \oplus_{i\in I} ! a_i . \overline{T}_i$.

Hereafter, we write $\leqslant_{\mathsf{a}}$ for the relation in [9] (abstracting away from carried types) which we recall below. An *asynchronous context* [9] is defined by

$$\mathcal{A} \coloneqq [\,]^n \;\mid\; \&_{i\in I} ? a_i . \mathcal{A}_i$$

We write $\mathcal{A}[\,]^{n\in N}$ to denote a context with holes indexed by elements of N and $\mathcal{A}[T_n]^{n\in N}$ to denote the same context when the hole $[\,]^n$ has been filled with T_n.

The predicate $\& \in T$ holds if it can be derived from the following rules:

$$\frac{}{\& \in \&_{i\in I} ? a_i . T_i} \qquad \frac{\forall i \in I \,:\, \& \in T_i}{\& \in \oplus_{i\in I} ! a_i . T_i} \qquad \frac{\& \in T}{\& \in \mathtt{rec}\, \mathbf{x}.T}$$

$\& \in T$ holds whenever T always eventually performs a receive action, i.e., it cannot loop on send actions only. It is the counterpart of the predicate $\mathtt{fin}(\_)$ for CFSMs, cf. Lemma 5.1.

Definition 5.2 ($\leqslant_{\mathsf{a}}$ [9])**.** *The asynchronous subtyping,* $\leqslant_{\mathsf{a}}$*, is the* largest *relation that contains the rules:*[1]

$$\frac{\forall i \in I \,:\, T_i \leqslant_{\mathsf{a}} U_i}{\oplus_{i\in I} !a_i.\, T_i \leqslant_{\mathsf{a}} \oplus_{i\in I\cup J} !a_i.\, U_i}\ [\textsc{sel}] \qquad \frac{\forall i \in I \,:\, T_i \leqslant_{\mathsf{a}} U_i}{\&_{i\in I\cup J} ?a_i.\, T_i \leqslant_{\mathsf{a}} \&_{i\in I} ?a_i.\, U_i}\ [\textsc{bra}]$$

$$\frac{\forall i \in I \,:\, T_i \leqslant_{\mathsf{a}} \mathcal{A}[U_i^n]^{n\in N} \quad \& \in T_i}{\oplus_{i\in I} !a_i.\, T_i \leqslant_{\mathsf{a}} \mathcal{A}[\oplus_{i\in I\cup J_n} !a_i.\, U_i^n]^{n\in N}}\ [\textsc{async}] \qquad \frac{}{\mathtt{end} \leqslant_{\mathsf{a}} \mathtt{end}}\ [\textsc{end}]$$

The double line in the rules indicates that the rules should be interpreted coinductively*. We are assuming an equi-recursive view of types.*

Rule [sel] lets the subtype make fewer selections than its supertype, while rule [bra] allows the subtype to offer more branches. Rule [async] allows safe permutations of actions, by which a protocol can be refined to maximise asynchrony without violating safety. Note that the *synchronous* subtyping $\leqslant_{\mathsf{s}}$ [11,19,20] is defined as Definition 5.2 without rule [async], hence $\leqslant_{\mathsf{s}} \subseteq \leqslant_{\mathsf{a}}$. In Fig. 1, $\mathcal{T}(M_s') \leqslant_{\mathsf{s}} \mathcal{T}(M_s)$, $\mathcal{T}(M_s') \leqslant_{\mathsf{a}} \mathcal{T}(M_s)$, and $\mathcal{T}(M_c) \leqslant_{\mathsf{a}} \mathcal{T}(M_c')$.

The correspondence between $\asymp$ (Definition 3.1) and $\leqslant_{\mathsf{a}}$ (Definition 5.2) can be understood as follows. Case (1) of Definition 3.1 corresponds to rule [end]. Case (2a) corresponds to rule [bra]. Case (3a) corresponds to rule [sel]. Cases (2b) and (3b) together correspond to rule [async].

Correspondences. We show that $\asymp$ on CFSMs and $\leqslant_{\mathsf{a}}$ on session types are equivalent, and, as a consequence, deciding whether two types are $\leqslant_{\mathsf{a}}$-related is undecidable. We first introduce a few auxiliary lemmas and definitions.

Lemma 5.1. *Let* $M = (Q, q_0, \delta)$ *and* T *be a session type.*

1. *For each* $q \in Q$*, if* $\mathtt{fin}(q)$*, then* $\& \in \mathcal{T}(Q, q, \delta)$*.*
2. *If* $\& \in T$ *and* $\mathcal{M}(T) = (\hat{Q}, q, \hat{\delta})$*, then* $\mathtt{fin}(q)$*.*
3. *If* $T = \mathcal{A}[\oplus_{i\in I} !a_i.\, U_i^n]^{n\in N}$ *then* $\& \in T$*.*

Lemma 5.1 states the relationship between $\& \in T$ and $\mathtt{fin}(\_)$ (cf. Sect. 2.2).

We write $\pi \in \mathcal{A}$ if π is a branch in the context $\mathcal{A}$. Formally, given $\mathcal{A}$ and $\pi \in \mathbb{A}^*$, we define the predicate $\pi \in \mathcal{A}$ as follows:

$$\pi \in \mathcal{A} \iff \begin{cases} \pi = \epsilon & \text{if } \mathcal{A} = [\,] \\ \pi = a_j \cdot \pi_j & \text{if } \mathcal{A} = \&_{i\in I} ?a_i.\, \mathcal{A}_i, \pi_j \in \mathcal{A}_j, \text{ with } j \in I \end{cases}$$

The next lemma shows that the $\leqslant_{\mathsf{a}}$-relation implies the $\asymp$-relation.

[1] Note that in [9] rule [async] has a redundant additional premise, $\& \in \mathcal{A}$, which is only used to make the application of the rules deterministic.

Lemma 5.2. *Let T and U be two session types, such that $\mathcal{M}(T) = (Q^T, q_0^T, \delta^T)$ and $\mathcal{M}(U) = (Q^U, q_0^U, \delta^U)$, then $T \leqslant_{\mathsf{a}} \mathcal{A}[\overline{U}] \implies \forall \pi \in \mathcal{A} : \pi \blacktriangleright q_0^T \asymp q_0^U$.*

The proof of Lemma 5.2 is by coinduction on the derivation of $\pi \blacktriangleright p \asymp q$. We use Lemma 5.1 to show that premise of rule [ASYNC] implies that $\mathtt{fin}(q_0^T)$ and $\mathtt{fin}(q_0^U)$ hold when necessary.

The next lemma shows that the $\asymp$-relation implies the $\leqslant_{\mathsf{a}}$-relation.

Lemma 5.3. *Let $M_i = (Q_i, q_{0_i}, \delta_i)$, $i \in \{1, 2\}$ and $\pi = a_1 \cdots a_k \in \mathbb{A}^*$, for all $p \in Q_1$ and $q \in Q_2$, $\pi \blacktriangleright p \asymp q \implies \mathcal{T}(Q_1, p, \delta_1) \leqslant_{\mathsf{a}} ?a_1 \cdots ?a_k.[\overline{\mathcal{T}(Q_2, q, \delta_2)}]$.*

The proof of Lemma 5.3 is by coinduction on the rules of Definition 5.2, using Lemma 5.1 to match the requirements of the respective relations.

We are now ready to state the final equivalence result.

Theorem 5.1. *The relations $\asymp$ and $\leqslant_{\mathsf{a}}$ are equivalent in the following sense:*

1. *Let T_1 and T_2 be two session types, then $T_1 \leqslant_{\mathsf{a}} \overline{T_2} \implies \mathcal{M}(T_1) \asymp \mathcal{M}(T_2)$.*
2. *Let M_1 and M_2 be two machines, then $M_1 \asymp M_2 \implies \mathcal{T}(M_1) \leqslant_{\mathsf{a}} \overline{\mathcal{T}(M_2)}$.*

Proof. (1) follows from Lemma 5.2, with $T_1 = T$, $T_2 = U$, and $\mathcal{A} = [\,]$. (2) follows from Lemma 5.3, with $\pi = \epsilon$, $p = q_{0_1}$, and $q = q_{0_2}$. □

A consequence of the correspondence between the two relations is that the $\asymp$-relation is transitive in the following sense:

Theorem 5.2. *If $M_1 \asymp \overline{M}$ and $M \asymp M_2$, then $M_1 \asymp M_2$.*

Proof. By Theorem 5.1 we have (1) $M_1 \asymp \overline{M} \iff M_1 \leqslant_{\mathsf{a}} M$ (2) $M \asymp M_2 \iff M \leqslant_{\mathsf{a}} \overline{M_2}$. Since $\leqslant_{\mathsf{a}}$ is transitive [10], we have $M_1 \leqslant_{\mathsf{a}} \overline{M_2}$. Thus, using Theorem 5.1 again, we have $M_1 \leqslant_{\mathsf{a}} \overline{M_2} \iff M_1 \asymp M_2$. □

As a consequence of Theorems 4.1 and 5.1, we have the undecidability of the asynchronous subtyping:

Theorem 5.3. (Undecidability of $\leqslant_{\mathsf{a}}$). *Given two session types T_1 and T_2, it is generally undecidable whether $T_1 \leqslant_{\mathsf{a}} T_2$ holds.*

We state the equivalence between $\asymp_s$ and $\leqslant_{\mathsf{s}}$, and the transitivity of $\asymp_s$.

Theorem 5.4. *The relations $\asymp_s$ and $\leqslant_{\mathsf{s}}$ are equivalent in the following sense:*

1. *Let T_1 and T_2 be two session types, then $T_1 \leqslant_{\mathsf{s}} \overline{T_2} \implies \mathcal{M}(T_1) \asymp_s \mathcal{M}(T_2)$.*
2. *Let M_1 and M_2 be two machines, then $M_1 \asymp_s M_2 \implies \mathcal{T}(M_1) \leqslant_{\mathsf{s}} \overline{\mathcal{T}(M_2)}$.*

Theorem 5.5. *If $M_1 \asymp_s \overline{M}$ and $M \asymp_s M_2$, then $M_1 \asymp_s M_2$.*

Theorem 5.1 together with the soundness and completeness of $\asymp$ wrt. safety in AD systems (Theorems 3.1 and 3.3) imply a tight relationship between $\leqslant_{\mathsf{a}}$ and session types corresponding to AD systems. A similar correspondence between $\leqslant_{\mathsf{s}}$ and HD systems exists, by Theorems 3.2, 3.4, and 5.4.

6 Conclusions and Related Work

We have introduced a new sub-class of CFSMs (AD), which includes HD, and a compatibility relation $\asymp$ (resp. $\asymp_s$) that is sound and complete wrt. safety within AD (resp. HD) and equivalent to asynchronous (resp. synchronous) subtyping. Our results in Sect. 4.1 suggest that $\asymp$ is a convenient basis for designing safety checking algorithms for some sub-classes of CFSMs. Given the results in the present paper, we plan to study bounded approximations of $\asymp$ that can be used for session typed applications. Such approximations would make asynchronous subtyping available for real-world programs and thus facilitate the integration of flexible session typing.

Related Work. The first (synchronous) subtyping for session types in the π-calculus was introduced in [19] and shown to be decidable in [20]. Its complexity was further studied in [26] which encodes synchronous subtyping as a model checking problem. The first version of asynchronous subtyping was introduced in [31] for multiparty session types and further studied in [28–30] for binary session types in the higher-order π-calculus. These works and [10] stated or conjectured the decidability of the relations. The technical report [5] (announced after the submission of the present paper) independently studied the undecidability of these relations. Note that the subtyping relation in [28,30] only differs from the one in [9,10] by the omission of the premise $\& \in T_i$ in rule [ASYNC]. This subtyping is not sound wrt. our definition of safety as it does not guarantee eventual reception [9,10]. We conjecture that it is sound and complete wrt. progress (either both machines are in a final state or one can eventually make a move) in (the full class of) CFSMs, hence it is also undecidable since progress corresponds to rejection of a word by a Turing machine, cf. Sect. 4.

The operational and denotational preciseness of (synchronous and asynchronous) subtyping for session types was studied in [9,10] where the authors give soundness and completeness of each subtyping through the set of π-calculus processes a type T can type. In this paper, we study the soundness and completeness of $\asymp$ (resp. $\asymp_s$) in CFSMs through AD (resp. HD) systems.

CFSMs have long been known to be Turing complete [4,17] even when restricted to deterministic machines without mixed states [21]. The first paper to relate formally CFSMs and session types was [14], which was followed by a series of work using CFSMs as session types [3,15,25]. The article [2] shows, in a similar fashion to [17], that the compliance of contracts based on asynchronous session types is undecidable. Here, we show that the encoding of [17] is indeed AD and that safety is equivalent to word acceptance by a Turing machine.

Acknowledgements. We thank A. Scalas, B. Toninho and G. Zavattaro for their comments on earlier versions of this paper, in particular G. Zavattaro for identifying the need for additional blank insertion cycles (in Fig. 3). This work is partially supported by EPSRC EP/K034413/1, EP/K011715/1, EP/L00058X/1, EP/N027833/1 and EP/N028201/1; and by EU FP7 612985 (UPSCALE).

References

1. Ancona, D., Bono, V., Bravetti, M., Campos, J., Castagna, G., Deniélou, P., Gay, S.J., Gesbert, N., Giachino, E., Hu, R., Johnsen, E.B., Martins, F., Mascardi, V., Montesi, F., Neykova, R., Ng, N., Padovani, L., Vasconcelos, V.T., Yoshida, N.: Behavioral types in programming languages. Found. Trends Program. Lang. **3**(2–3), 95–230 (2016)
2. Bartoletti, M., Scalas, A., Tuosto, E., Zunino, R.: Honesty by typing. Log. Meth. Comput. Sci. **12**(4) (2016)
3. Bocchi, L., Lange, J., Yoshida, N.: Meeting deadlines together. In: CONCUR 2015, pp. 283–296 (2015)
4. Brand, D., Zafiropulo, P.: On communicating finite-state machines. J. ACM **30**(2), 323–342 (1983)
5. Bravetti, M., Carbone, M., Zavattaro, G.: Undecidability of asynchronous session subtyping. CoRR, abs/1611.05026 (2016)
6. Carbone, M., Honda, K., Yoshida, N.: Structured communication-centred programming for web services. In: Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 2–17. Springer, Heidelberg (2007). doi:10.1007/978-3-540-71316-6_2
7. Carbone, M., Honda, K., Yoshida, N.: Structured communication-centered programming for web services. ACM Trans. Program. Lang. Syst. **34**(2), 8 (2012)
8. Cécé, G., Finkel, A.: Verification of programs with half-duplex communication. Inf. Comput. **202**(2), 166–190 (2005)
9. Chen, T.-C., Dezani-Ciancaglini, M., Scalas, A., Yoshida, N.: On the preciseness of subtyping in session types. LMCS (2016)
10. Chen, T.-C., Dezani-Ciancaglini, M., Yoshida, N.: On the preciseness of subtyping in session types. In: PPDP 2014, pp. 146–135. ACM Press (2014)
11. Demangeon, R., Honda, K.: Full abstraction in a subtyped pi-calculus with linear types. In: Katoen, J.-P., König, B. (eds.) CONCUR 2011. LNCS, vol. 6901, pp. 280–296. Springer, Heidelberg (2011). doi:10.1007/978-3-642-23217-6_19
12. Demangeon, R., Yoshida, N.: On the expressiveness of multiparty sessions. In: Harsha, P., Ramalingam, G. (eds.) FSTTCS 2015. LIPIcs, vol. 45, pp. 560–574. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2015)
13. Deniélou, P., Yoshida, N.: Buffered communication analysis in distributed multiparty sessions. In: CONCUR 2010, pp. 343–357 (2010)
14. Deniélou, P.-M., Yoshida, N.: Multiparty session types meet communicating automata. In: Seidl, H. (ed.) ESOP 2012. LNCS, vol. 7211, pp. 194–213. Springer, Heidelberg (2012). doi:10.1007/978-3-642-28869-2_10
15. Deniélou, P., Yoshida, N.: Multiparty compatibility in communicating automata: characterisation and synthesis of global session types. In: ICALP 2013, pp. 174–186 (2013)
16. Dezani-Ciancaglini, M., Ghilezan, S., Jaksic, S., Pantovic, J., Yoshida, N.: Denotational and operational preciseness of subtyping: a roadmap. In: Theory and Practice of Formal Methods - Essays Dedicated to Frank de Boer on the Occasion of His 60th Birthday, pp. 155–172 (2016)
17. Finkel, A., McKenzie, P.: Verifying identical communicating processes is undecidable. Theor. Comput. Sci. **174**(1–2), 217–230 (1997)
18. Gay, S.J.: Subtyping supports safe session substitution. In: A List of Successes That Can Change the World - Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday, pp. 95–108 (2016)

19. Gay, S., Hole, M.: Types and subtypes for client-server interactions. In: Swierstra, S.D. (ed.) ESOP 1999. LNCS, vol. 1576, pp. 74–90. Springer, Heidelberg (1999). doi:10.1007/3-540-49099-X_6
20. Gay, S.J., Hole, M.: Subtyping for session types in the pi calculus. Acta Inf. **42**(2–3), 191–225 (2005)
21. Gouda, M.G., Manning, E.G., Yu, Y.: On the progress of communications between two finite state machines. Inf. Control **63**(3), 200–216 (1984)
22. Honda, K., Vasconcelos, V.T., Kubo, M.: Language primitives and type discipline for structured communication-based programming. In: Hankin, C. (ed.) ESOP 1998. LNCS, vol. 1381, pp. 122–138. Springer, Heidelberg (1998). doi:10.1007/BFb0053567
23. Hu, R., Yoshida, N.: Hybrid session verification through endpoint API generation. In: FASE 2016, pp. 401–418 (2016)
24. Hüttel, H., Lanese, I., Vasconcelos, V.T., Caires, L., Carbone, M., Deniélou, P., Mostrous, D., Padovani, L., Ravara, A., Tuosto, E., Vieira, H.T., Zavattaro, G.: Foundations of session types and behavioural contracts. ACM Comput. Surv. **49**(1), 3 (2016)
25. Lange, J., Tuosto, E., Yoshida, N.: From communicating machines to graphical choreographies. In: POPL 2015, pp. 221–232 (2015)
26. Lange, J., Yoshida, N.: Characteristic formulae for session types. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 833–850. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49674-9_52
27. Lange, J., Yoshida, N.: On the undecidability of asynchronous session subtyping (with appendices). Technical report 2016/9, Imperial College London (2016). https://www.doc.ic.ac.uk/research/technicalreports/2016/DTRS16-9.pdf
28. Mostrous, D.: Session types in concurrent calculi: higher-order processes and objects. PhD thesis, Imperial College London, November 2009
29. Mostrous, D., Yoshida, N.: Session-based communication optimisation for higher-order mobile processes. In: Curien, P.-L. (ed.) TLCA 2009. LNCS, vol. 5608, pp. 203–218. Springer, Heidelberg (2009). doi:10.1007/978-3-642-02273-9_16
30. Mostrous, D., Yoshida, N.: Session typing and asynchronous subtyping for the higher-order π-calculus. Inf. Comput. **241**, 227–263 (2015)
31. Mostrous, D., Yoshida, N., Honda, K.: Global principal typing in partially commutative asynchronous sessions. In: Castagna, G. (ed.) ESOP 2009. LNCS, vol. 5502, pp. 316–332. Springer, Heidelberg (2009). doi:10.1007/978-3-642-00590-9_23
32. Ng, N., Figueiredo Coutinho, J.G., Yoshida, N.: Protocols by default. In: Franke, B. (ed.) CC 2015. LNCS, vol. 9031, pp. 212–232. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46663-6_11
33. Ng, N., Yoshida, N., Honda, K.: Multiparty session C: safe parallel programming with message optimisation. In: Furia, C.A., Nanz, S. (eds.) TOOLS 2012. LNCS, vol. 7304, pp. 202–218. Springer, Heidelberg (2012). doi:10.1007/978-3-642-30561-0_15
34. Takeuchi, K., Honda, K., Kubo, M.: An interaction-based language and its typing system. In: Halatsis, C., Maritsas, D., Philokyprou, G., Theodoridis, S. (eds.) PARLE 1994. LNCS, vol. 817, pp. 398–413. Springer, Heidelberg (1994). doi:10.1007/3-540-58184-7_118
35. Yoshida, N., Vasconcelos, V., Paulino, H., Honda, K.: Session-based compilation framework for multicore programming. In: Boer, F.S., Bonsangue, M.M., Madelaine, E. (eds.) FMCO 2008. LNCS, vol. 5751, pp. 226–246. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04167-9_12

Lambda Calculus and Constructive Proof

A Lambda-Free Higher-Order Recursive Path Order

Jasmin Christian Blanchette[1,2,3], Uwe Waldmann[3], and Daniel Wand[3,4]

[1] Vrije Universiteit Amsterdam, Amsterdam, The Netherlands
[2] Inria Nancy – Grand Est, Villers-lès-Nancy, France
[3] Max-Planck-Institut für Informatik, Saarbrücken, Germany
[4] Institut für Informatik, Technische Universität München, Munich, Germany

Abstract. We generalize the recursive path order (RPO) to higher-order terms without λ-abstraction. This new order fully coincides with the standard RPO on first-order terms also in the presence of currying, distinguishing it from previous work. It has many useful properties, including well-foundedness, transitivity, stability under substitution, and the subterm property. It appears promising as the basis of a higher-order superposition calculus.

1 Introduction

Most automatic reasoning tools are restricted to first-order formalisms, even though many proof assistants and specification languages are higher-order. Translations bridge the gap, but they usually have a cost. Thus, a recurrent question in our field is, *Which first-order methods can be gracefully extended to a higher-order setting?* By "gracefully," we mean that the higher-order extension of the method is as powerful as its first-order counterpart on the first-order portions of the input.

The distinguishing features of higher-order terms are that (1) they support currying, meaning that an n-ary function may be applied to fewer than n arguments, (2) variables can be applied, and (3) λ-abstractions, written $\lambda x.t_x$, can be used to specify anonymous functions $x \mapsto t_x$. Iterated applications are written without parentheses or commas, as in $\mathsf{f}\,\mathsf{a}\,\mathsf{b}$. Many first-order proof calculi have been extended to higher-order logic, including resolution and tableaux, but so far there exists no sound and complete higher-order version of superposition [29], where completeness is considered with respect to Henkin semantics [4,18]. Together with CDCL(T) [17], superposition is one of the leading proof calculi for classical first-order logic with equality.

To prune the search space, superposition depends on a term order, which is fixed in advance of the proof attempt. For example, from $\mathsf{p}(\mathsf{a})$ and $\neg\, \mathsf{p}(x) \vee \mathsf{p}(\mathsf{f}(x))$, resolution helplessly derives infinitely many clauses of the form $\mathsf{p}(\mathsf{f}^i(\mathsf{a}))$, whereas for superposition the literal $\mathsf{p}(\mathsf{f}(x))$ is maximal in its clause and blocks all inferences. To work with superposition, the order must fulfill many requirements, including compatibility with contexts, stability under substitution, and

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 461–479, 2017.
DOI: 10.1007/978-3-662-54458-7_27

totality on ground (variable-free) terms. The lexicographic path order (LPO) and the Knuth–Bendix order (KBO) [3] both fulfill the requirements. LPO is a special case of the recursive path order (RPO), which also subsumes the multiset path order [38]. Suitable generalizations of LPO and KBO appear to be crucial ingredients of a future higher-order superposition prover.

A simple technique to support currying and applied variables is to make all symbols nullary and to represent application by a distinguished binary symbol $@$. Thus, the higher-order term $\mathsf{f}\,(x\,\mathsf{f})$ is translated to $@(\mathsf{f}, @(x, \mathsf{f}))$, which can be processed by first-order methods. We call this the *applicative encoding*. As for λ-abstractions, in many settings they can be avoided using λ-lifting [21] or SK combinators [36]. A drawback of the applicative encoding is that argument tuples cannot be compared using different methods for different function symbols. The use of an application symbol also weakens the order in other ways [25, Sect. 2.3.1]. Hybrid schemes have been proposed to strengthen the encoding: If a function f always occurs with at least k arguments, these can be passed directly in an uncurried style—e.g., $@(\mathsf{f}(\mathsf{a}, \mathsf{b}), x)$. However, this relies on a closed-world assumption—namely, that all terms that will ever be compared arise in the input problem. This is at odds with the need for complete higher-order proof calculi to synthesize arbitrary terms during proof search [4], in which a symbol f may be applied to fewer arguments than anywhere in the problem. A scheme by Hirokawa et al. [19] circumvents this issue but requires additional symbols and rewrite rules.

Versions of RPO tailored for higher-order terms are described in the literature, including Lifantsev and Bachmair's LPO on λ-free higher-order terms [27], Jouannaud and Rubio's higher-order RPO (HORPO) [23], Kop and van Raamsdonk's iterative HORPO [26], the HORPO extension with polynomial interpretation orders by Bofill et al. [12], and the computability path order by Blanqui et al. [10], also a variant of HORPO. All of these combine uncurrying and currying: They distinguish between *functional* arguments, which are passed directly as a tuple to a function, and *applicative* arguments, which are optional. Coincidence with the standard RPO on first-order terms is achieved only for uncurried functions. Techniques to automatically curry or uncurry functions have been developed, but they rely on the closed-world assumption. Moreover, the orders all lack totality on ground terms; the HORPO variants also lack the subterm property, and only their (noncomputable) transitive closure is transitive.

We introduce a new "graceful" order $>_{\mathsf{ho}}$ for untyped λ-free higher-order terms (Sect. 3). It generalizes the first-order RPO along two main axes: (1) It relies on a higher-order notion of subterm; (2) it supports terms with applied variables—e.g., $x\,\mathsf{b} >_{\mathsf{ho}} x\,\mathsf{a}$ if $\mathsf{b} \succ \mathsf{a}$ according to the underlying precedence $\succ$ on symbols. The order is parameterized by a family of abstractly specified extension operators indexed by function symbols, allowing lexicographic, multiset, and other extension operators. An optimized variant, $>_{\mathsf{oh}}$, coincides with $>_{\mathsf{ho}}$ under a reasonable assumption on the extension operator. For comparison, we also present the first-order RPO $>_{\mathsf{fo}}$ and its composition $>_{\mathsf{ap}}$ with the applicative encoding, both recast to our abstract framework.

The λ-free fragment is useful in its own right and constitutes a stepping stone towards full higher order. Our new order operates exclusively on curried functions while coinciding with the standard RPO on first-order terms. This was considered impossible by Lifantsev and Bachmair [27]:

> Pairs, or more generally tuples, allow one to compare the arguments of different functions with greater flexibility. For instance, the arguments of one function may be compared lexicographically, whereas in other cases comparison may be based on the multisets of arguments. ... But since function symbols are much more decoupled from their arguments in a higher-order setting than in a first-order setting, the information needed for different argument-comparison methods would be lost if one, say, just curried all functions.

The order $>_{\mathsf{ho}}$ enjoys many useful properties (Sect. 4). One property that is missing is compatibility with a specific type of higher-order context: If $s' >_{\mathsf{ho}} s$, it is still possible that $s'\,t \not>_{\mathsf{ho}} s\,t$. For example, if $\mathsf{g} \succ \mathsf{f} \succ \mathsf{b} \succ \mathsf{a}$, then $\mathsf{f}\,(\mathsf{g}\,\mathsf{a}) >_{\mathsf{ho}} \mathsf{g}$ by the subterm property, but $\mathsf{f}\,(\mathsf{g}\,\mathsf{a})\,\mathsf{b} <_{\mathsf{ho}} \mathsf{g}\,\mathsf{b}$ by coincidence with the first-order RPO [35]. Nonetheless, we expect the order to be usable for λ-free higher-order superposition, at the cost of some complications [13]. The proofs of the properties were carried out in a proof assistant, Isabelle/HOL [31], and are publicly available [7]. Informal proofs are included in this paper and in a technical report [8].

Beyond superposition, the order can also be employed to prove termination of higher-order term rewriting systems. Because it treats all functions as curried, it differs from the other higher-order RPOs on many examples (Sect. 5), thereby enriching the portfolio of methods available to termination provers.

Conventions. We fix a set $\mathcal{V}$ of *variables* with typical elements x, y. A higher-order signature consists of a nonempty set Σ of (function) *symbols* $\mathsf{a}, \mathsf{b}, \mathsf{f}, \mathsf{g}, \mathsf{h}, \ldots$. Untyped λ-free higher-order (Σ-)*terms* $s, t, u \in \mathcal{T}_\Sigma\ (= \mathcal{T})$ are defined inductively by the grammar $s ::= x \mid \mathsf{f} \mid t\,u$. These are isomorphic to applicative terms [24].

A term of the form tu is called an *application*. Non-application terms $\zeta, \xi \in \Sigma \uplus \mathcal{V}$ are called *heads*. Terms can be decomposed in a unique way as a head applied to zero or more arguments: $\zeta\ s_1\ \ldots\ s_m$. This view corresponds to the first-order, uncurried syntax $\zeta(s_1, \ldots, s_m)$, except that ζ may always be a variable.

The *size* $|s|$ of a term is the number of grammar rule applications needed to construct it. The set of variables occurring in s is written $\mathit{vars}(s)$. The set of *subterms* of a term s always contains s; for applications $t\,u$, it also includes all the subterms of t and u.

A *first-order* signature Σ extends a higher-order signature by associating an *arity* with each symbol belonging to Σ. A *first-order* term is a term in which variables are unapplied and symbols are applied to the number of arguments specified by their arity. For consistency, we will use a curried syntax for first-order terms.

2 Extension Orders

Orders such as RPO depend on extension operators to recurse through tuples of arguments. The literature is mostly concerned with the lexicographic and multiset orders [3,38]. We favor an abstract treatment that formulates requirements on the extension operators. Beyond its generality, this approach emphasizes the complications arising from the higher-order setting.

Let $A^* = \bigcup_{i=0}^{\infty} A^i$ be the set of tuples (or finite lists) of arbitrary length whose components are drawn from a set A. We write its elements as $(a_1, \ldots, a_m)$, where $m \geq 0$, or simply $\bar{a}$. The empty tuple is written $()$. Singleton tuples are identified with elements of A. The number of components of a tuple $\bar{a}$ is written $|\bar{a}|$. Given an m-tuple $\bar{a}$ and an n-tuple $\bar{b}$, we denote by $\bar{a} \cdot \bar{b}$ the $(m+n)$-tuple consisting of the concatenation of $\bar{a}$ and $\bar{b}$.

Given a function $h : A \to A$, we let $h(\bar{a})$ stand for the componentwise application of h to $\bar{a}$. Abusing notation, we sometimes use a tuple where a set or multiset is expected, ignoring the extraneous structure. Moreover, since all our functions are curried, we write $\zeta\ \bar{s}$ for a curried application $\zeta\ s_1\ \ldots\ s_m$, without risk of ambiguity.

Given a relation $>$, we write $<$ for its inverse (i.e., $a < b \iff b > a$) and $\geq$ for its reflexive closure (i.e., $b \geq a \iff b > a \vee b = a$). A (strict) partial order is a relation that is irreflexive (i.e., $a \not> a$) and transitive (i.e., $c > b \wedge b > a \Longrightarrow c > a$). A (strict) total order is a partial order that satisfies totality (i.e., $b \geq a \vee a > b$). A relation $>$ is well founded if and only if there exists no infinite chain of the form $a_0 > a_1 > \cdots$.

Let $\gg \subseteq (A^*)^2$ be a family of relations indexed by a relation $> \subseteq A^2$. For example, $\gg$ could be the lexicographic or multiset extension of $>$. The following properties are essential for all the orders defined later, whether first- or higher-order:

X1. *Monotonicity*: $\bar{b} \gg_1 \bar{a}$ implies $\bar{b} \gg_2 \bar{a}$ if $b >_1 a$ implies $b >_2 a$ for all a, b;
X2. *Preservation of stability*:
$\bar{b} \gg \bar{a}$ implies $h(\bar{b}) \gg h(\bar{a})$ if $b > a$ implies $h(b) > h(a)$ for all a, b;
X3. *Preservation of transitivity*: $\gg$ is transitive if $>$ is transitive;
X4. *Preservation of irreflexivity*: $\gg$ is irreflexive if $>$ is irreflexive and transitive;
X5. *Preservation of well-foundedness*: $\gg$ is well founded if $>$ is well founded;
X6. *Compatibility with tuple contexts*: $b > a$ implies $\bar{c} \cdot b \cdot \bar{d} \gg \bar{c} \cdot a \cdot \bar{d}$.

Because the relation $>$ will depend on $\gg$ for its definition, we cannot assume outright that it is a partial order, a fine point that is sometimes overlooked [38, Sect. 6.4.2].

The remaining properties of $\gg$ will be required only by some of the orders or for some optional properties of $>$:

X7. *Preservation of totality*: $\gg$ is total if $>$ is total;
X8. *Compatibility with prepending*: $\bar{b} \gg \bar{a}$ implies $a \cdot \bar{b} \gg a \cdot \bar{a}$;
X9. *Compatibility with appending*: $\bar{b} \gg \bar{a}$ implies $\bar{b} \cdot a \gg \bar{a} \cdot a$;
X10. *Minimality of empty tuple*: $a \gg ()$.

We now define the extension operators and study their properties. All of them are also defined for tuples of different lengths.

Definition 1. The *lexicographic extension* $\gg^{\mathsf{lex}}$ of the relation $>$ is defined recursively by $() \not\gg^{\mathsf{lex}} \bar{a}$, $b \cdot \bar{b} \gg^{\mathsf{lex}} ()$, and $b \cdot \bar{b} \gg^{\mathsf{lex}} a \cdot \bar{a} \Leftrightarrow b > a \vee b = a \wedge \bar{b} \gg^{\mathsf{lex}} \bar{a}$.

The reverse, or right-to-left, lexicographic extension is defined analogously. Both operators lack the essential property X5. In addition, the left-to-right version lacks X9, and the right-to-left version lacks X8. The other properties are straightforward to prove.

Definition 2. The *length-lexicographic extension* $\gg^{\mathsf{llex}}$ of the relation $>$ is defined by $\bar{b} \gg^{\mathsf{llex}} \bar{a} \Leftrightarrow |\bar{b}| > |\bar{a}| \vee |\bar{b}| = |\bar{a}| \wedge \bar{b} \gg^{\mathsf{lex}} \bar{a}$.

The length-lexicographic extension and its right-to-left counterpart satisfy all of the properties listed above. We can also apply arbitrary permutations on same-length tuples before comparing them lexicographically; however, the resulting operators generally fail to satisfy properties X8 and X9.

Definition 3. The *multiset extension* $\gg^{\mathsf{ms}}$ of the relation $>$ is defined by $\bar{b} \gg^{\mathsf{ms}} \bar{a} \Leftrightarrow \exists Y, X.\ \emptyset \neq Y \subseteq \bar{b} \wedge \bar{a} = (\bar{b} - Y) \uplus X \wedge \forall x \in X.\ \exists y \in Y.\ y > x$, where X, Y range over multisets, the tuples $\bar{a}, \bar{b}$ are implicitly converted to multisets, and $\uplus$ denotes multiset sum (the sum of the multiplicity functions).

The multiset extension, due to Dershowitz and Manna [16], satisfies all properties except X7. Huet and Oppen [20] give an alternative formulation that is equivalent for partial orders $>$ but exhibits subtle differences if $>$ is an arbitrary relation. In particular, the Huet–Oppen order does not satisfy property X3.

Finally, we consider the componentwise extension of relations to pairs of tuples of the same length. For partial orders $>$, this order underapproximates any extension that satisfies properties X3 and X6. It also satisfies all properties except X7.

Definition 4. The *componentwise extension* $\gg^{\mathsf{cw}}$ of the relation $>$ is defined so that $(b_1, \ldots, b_n) \gg^{\mathsf{cw}} (a_1, \ldots, a_m)$ if and only if $m = n$, $b_1 \geq a_1$, $\ldots$, $b_m \geq a_m$, and $b_i > a_i$ for some $i \in \{1, \ldots, m\}$.

3 Term Orders

This section presents four orders: the standard first-order RPO (Sect. 3.1), the applicative RPO (Sect. 3.2), our new λ-free higher-order RPO (Sect. 3.3), and an optimized variant of our new RPO (Sect. 3.4).

3.1 The Standard First-Order RPO

The following definition is close to Zantema's formulation [38, Definition 6.4.4] but adapted to our setting. With three rules instead of four, it is more concise than Baader and Nipkow's formulation of LPO [3, Definition 5.4.12] and lends itself better to a higher-order generalization.

Definition 5. Let $\succ$ be a well-founded total order on Σ, and let $\gg^{\mathsf{f}} \subseteq (\mathcal{T}^*)^2$ be a family of relations indexed by $> \subseteq \mathcal{T}^2$ and by $\mathsf{f} \in \Sigma$ and satisfying properties X1–X6. The induced *recursive path order* $>_{\mathsf{fo}}$ on first-order Σ-terms is defined inductively so that $t >_{\mathsf{fo}} s$ if any of the following conditions is met, where $t = \mathsf{g}\,\bar{t}$:

F1. $t' \geq_{\mathsf{fo}} s$ for some term $t' \in \bar{t}$;
F2. $s = \mathsf{f}\,\bar{s}$, $\mathsf{g} \succ \mathsf{f}$, and $\mathit{chkargs}(t, \bar{s})$;
F3. $s = \mathsf{f}\,\bar{s}$, $\mathsf{f} = \mathsf{g}$, $\bar{t} \gg^{\mathsf{f}}_{\mathsf{fo}} \bar{s}$, and $\mathit{chkargs}(t, \bar{s})$.

The auxiliary predicate $\mathit{chkargs}(t, \bar{s})$ is true if and only if $t >_{\mathsf{fo}} s'$ for all terms $s' \in \bar{s}$. The inductive definition is legitimate by the monotonicity of $\gg^{\mathsf{f}}$ (property X1).

RPO is a compromise between two design goals. On the one hand, rules F2 and F3, which form the core of the order, attempt to perform a comparison of two terms by first looking at their heads, proceeding recursively to break ties. On the other hand, rule F1 ensures that terms are larger than their proper subterms and, transitively, larger than terms smaller than these. The *chkargs* predicate prevents the application of F2 and F3 when F1 is applicable in the other direction, ensuring irreflexivity.

The more recent literature defines RPO somewhat differently: Precision is improved by replacing recursive calls to $\geq_{\mathsf{fo}}$ with a nonstrict quasiorder $\gtrsim_{\mathsf{fo}}$ and by exploiting a generalized multiset extension [14,33]. These extensions are useful but require substantial duplication in the definitions and the proofs, without yielding much new insight into orders for higher-order terms.

3.2 The Applicative RPO

Applicative orders are built by encoding applications using a binary symbol $@$ and by employing a first-order term order. For RPO, the precedence $\succ$ must be extended to consider $@$. A natural choice is to make $@$ the least element of $\succ$. Because $@$ is the only symbol that may be applied, $\gg^{@}$ is the only member of the $\gg$ family that is relevant. This means that it is impossible to use the lexicographic extension for some functions and the multiset extension for others.

Definition 6. Let Σ be a higher-order signature, and let $\Sigma' = \Sigma \uplus \{@\}$ be a first-order signature in which all symbols belonging to Σ are assigned arity 0 and $@$ is assigned arity 2. The *applicative encoding* $[\![\,]\!] : \mathcal{T}_\Sigma \to \mathcal{T}_{\Sigma'}$ is defined recursively by the equations $[\![\zeta]\!] = \zeta$ and $[\![s\,t]\!] = @\,[\![s]\!]\,[\![t]\!]$.

Assuming that $@$ has the lowest precedence, the composition of the first-order RPO with the encoding $[\![\,]\!]$ can be formulated directly as follows.

Definition 7. Let $\succ$ be a well-founded total order on Σ, and let $\gg \subseteq (\mathcal{T}^*)^2$ be a family of relations indexed by $> \subseteq \mathcal{T}^2$ and satisfying properties X1–X6. The induced *applicative recursive path order* $>_{\mathsf{ap}}$ on higher-order Σ-terms is defined inductively so that $t >_{\mathsf{ap}} s$ if any of the following conditions is met:

A1. $t = t_1\, t_2$ and either $t_1 \geq_{\mathsf{ap}} s$ or $t_2 \geq_{\mathsf{ap}} s$ (or both);
A2. $t = \mathsf{g} \succ \mathsf{f} = s$;
A3. $t = \mathsf{g}$, $s = s_1\, s_2$, and $\mathit{chkargs}(t, s_1, s_2)$;
A4. $t = t_1\, t_2$, $s = s_1\, s_2$, $(t_1, t_2) \gg_{\mathsf{ap}} (s_1, s_2)$, and $\mathit{chkargs}(t, s_1, s_2)$.

The predicate $\mathit{chkargs}(t, s_1, s_2)$ is true if and only if $t >_{\mathsf{ap}} s_1$ and $t >_{\mathsf{ap}} s_2$.

3.3 A Graceful Higher-Order RPO

Our new "graceful" higher-order RPO is much closer to the first-order RPO than the applicative RPO. It reintroduces the symbol-indexed family of extension operators and consists of three rules H1–H3 corresponding to F1–F3.

The order relies on a mapping ghd from variables to nonempty sets of possible ground heads that may arise when instantiating the variables. This mapping is extended to symbols f by taking $\mathit{ghd}(\mathsf{f}) = \{\mathsf{f}\}$. A substitution $\sigma : \mathcal{V} \to \mathcal{T}$ is said to *respect* the ghd mapping if for all variables x, we have $\mathit{ghd}(\zeta) \subseteq \mathit{ghd}(x)$ whenever $x\sigma = \zeta\, \bar{s}$. This mapping allows us to restrict instantiations, typically based on a typing discipline, and thereby increase the applicability of rules H2 and especially H3. Precedences $\succ$ are extended to variables by taking $y \succ x \iff \forall \mathsf{g} \in \mathit{ghd}(y), \mathsf{f} \in \mathit{ghd}(x).\ \mathsf{g} \succ \mathsf{f}$.

Definition 8. Let $\succ$ be a well-founded total order on Σ, let $\gg^{\mathsf{f}} \subseteq (\mathcal{T}^*)^2$ be a family of relations indexed by $> \subseteq \mathcal{T}^2$ and by $\mathsf{f} \in \Sigma$ and satisfying properties X1–X6 and X8, and let $\mathit{ghd} : \mathcal{V} \to \mathcal{P}(\Sigma) - \{\emptyset\}$. The induced *graceful recursive path order* $>_{\mathsf{ho}}$ on higher-order Σ-terms is defined inductively so that $t >_{\mathsf{ho}} s$ if any of the following conditions is met, where $s = \zeta\, \bar{s}$ and $t = \xi\, \bar{t}$:

H1. $t = t_1\, t_2$ and either $t_1 \geq_{\mathsf{ho}} s$ or $t_2 \geq_{\mathsf{ho}} s$ (or both);
H2. $\xi \succ \zeta$, $\mathit{vars}(t) \supseteq \mathit{vars}(\zeta)$, and $\mathit{chksubs}(t, s)$;
H3. $\xi = \zeta$, $\bar{t} \gg^{\mathsf{f}}_{\mathsf{ho}} \bar{s}$ for all symbols $\mathsf{f} \in \mathit{ghd}(\zeta)$, and $\mathit{chksubs}(t, s)$.

The predicate $\mathit{chksubs}(t, s)$ is true if and only if term s is a head or an application of the form $s_1\, s_2$ with $t >_{\mathsf{ho}} s_1$ and $t >_{\mathsf{ho}} s_2$.

There are two main novelties compared with $>_{\mathsf{fo}}$. First, rule H1 and the $\mathit{chksubs}$ predicate traverse subterms in a genuinely higher-order fashion. Second, rules H2 and H3 can compare terms with variable heads.

Property X8, compatibility with prepending, is necessary to ensure stability under substitution: If $x\, \mathsf{b} >_{\mathsf{ho}} x\, \mathsf{a}$, we want $\mathsf{f}\, \bar{s}\, \mathsf{b} >_{\mathsf{ho}} \mathsf{f}\, \bar{s}\, \mathsf{a}$ to hold as well.

Example 9. It is instructive to contrast our new order with the applicative order by studying a few small examples. Let $\mathsf{h} \succ \mathsf{g} \succ \mathsf{f} \succ \mathsf{b} \succ \mathsf{a}$, let $\gg$ be the length-lexicographic extension (which degenerates to the plain lexicographic

extension for $>_{\mathsf{ap}}$), and let $\mathit{ghd}(x) = \Sigma$ for all variables x. Sect. 1 already presented a case where $>_{\mathsf{ho}}$ and $>_{\mathsf{ap}}$ disagree: $\mathsf{g}\,\mathsf{b} >_{\mathsf{ho}} \mathsf{f}\,(\mathsf{g}\,\mathsf{a})\,\mathsf{b}$ but $\mathsf{g}\,\mathsf{b} <_{\mathsf{ap}} \mathsf{f}\,(\mathsf{g}\,\mathsf{a})\,\mathsf{b}$. Other disagreements include

$$\mathsf{g}\,\mathsf{f} >_{\mathsf{ho}} \mathsf{f}\,\mathsf{g}\,\mathsf{f} \qquad \mathsf{g}\,\mathsf{f} >_{\mathsf{ho}} \mathsf{f}\,\mathsf{g}\,(\mathsf{f}\,\mathsf{g}) \qquad \mathsf{g}\,\mathsf{g} >_{\mathsf{ho}} \mathsf{f}\,\mathsf{g}\,\mathsf{g} \qquad \mathsf{g}\,(\mathsf{f}\,\mathsf{h}) >_{\mathsf{ho}} \mathsf{f}\,\mathsf{h}\,(\mathsf{f}\,\mathsf{h})$$

and $\mathsf{g}\,\mathsf{g}\,\mathsf{g}\,(\mathsf{f}\,(\mathsf{g}\,(\mathsf{g}\,\mathsf{g}\,\mathsf{g}))) >_{\mathsf{ho}} \mathsf{g}\,(\mathsf{g}\,\mathsf{g}\,\mathsf{g})\,(\mathsf{g}\,\mathsf{g}\,\mathsf{g})$. For all of these, the core rules H2 and H3 are given room for maneuver, whereas $>_{\mathsf{ap}}$ must consider subterms using A1. In the presence of variables, some terms are comparable only with $>_{\mathsf{ho}}$ or only with $>_{\mathsf{ap}}$:

$$\mathsf{g}\,x >_{\mathsf{ho}} \mathsf{f}\,x\,x \qquad \mathsf{g}\,x >_{\mathsf{ho}} \mathsf{f}\,x\,\mathsf{g} \qquad \mathsf{f}\,x\,y >_{\mathsf{ap}} x\,y \qquad x\,\mathsf{f}\,(x\,\mathsf{f}) >_{\mathsf{ap}} \mathsf{f}\,x$$

To apply rule A4 on the first example, we would need $(\mathsf{g}, x) \gg_{\mathsf{ap}}^{\mathsf{lex}} (\mathsf{f}\,x, x)$, but the term g cannot be larger than $\mathsf{f}\,x$ since it does not contain x. The last two examples reveal that the applicative order tends to be stronger when either side is a variable applied to some arguments—at least when ghd is not restricting the variable instantiations.

3.4 An Optimized Variant of the Graceful Higher-Order RPO

The higher-order term $\mathsf{f}\,\mathsf{a}\,\mathsf{b}$ has four proper subterms: a, b, f, and $\mathsf{f}\,\mathsf{a}$. In contrast, the corresponding first-order term, traditionally written $\mathsf{f}(\mathsf{a}, \mathsf{b})$, has only the arguments a and b as proper subterms. In general, a term of size k has up to $k-1$ distinct proper subterms in a higher-order sense but only half as many in a first-order sense. By adding a reasonable requirement on the extension operator, we can avoid this factor-of-2 penalty when computing the order.

Definition 10. Let $\succ$ be a well-founded total order on Σ, let $\gg^{\mathsf{f}} \subseteq (\mathcal{T}^*)^2$ be a family of relations indexed by $> \subseteq \mathcal{T}^2$ and by $\mathsf{f} \in \Sigma$ and satisfying properties X1–X6, X8, and X10, and let $\mathit{ghd} : \mathcal{V} \to \mathcal{P}(\Sigma) - \{\emptyset\}$. The induced *optimized graceful recursive path order* $>_{\mathsf{oh}}$ on higher-order Σ-terms is defined inductively so that $t >_{\mathsf{oh}} s$ if any of the following conditions is met, where $s = \zeta\,\bar{s}$ and $t = \xi\,\bar{t}$:

O1. $t' \geq_{\mathsf{oh}} s$ for some term $t' \in \bar{t}$;
O2. $\xi \succ \zeta$, $\mathit{vars}(t) \supseteq \mathit{vars}(\zeta)$, and $\mathit{chkargs}(t, \bar{s})$;
O3. $\xi = \zeta$, $\bar{t} \gg_{\mathsf{oh}}^{\mathsf{f}} \bar{s}$ for all symbols $\mathsf{f} \in \mathit{ghd}(\zeta)$, and $\mathit{chkargs}(t, \bar{s})$.

The predicate $\mathit{chkargs}(t, \bar{s})$ is true if and only if $t >_{\mathsf{oh}} s'$ for all terms $s' \in \bar{s}$.

The optimized $>_{\mathsf{oh}}$ depends on the same parameters as $>_{\mathsf{ho}}$ except that it additionally requires minimality of the empty tuple (property X10). In conjunction with compatibility with prepending (X8), this property ensures that $\bar{a} \cdot a \gg^{\mathsf{f}} \bar{a}$. As a result, $\mathsf{f}\,\bar{s}\,s$ is greater than its subterm $\mathsf{f}\,\bar{s}$, relieving rule O1 from having to consider such subterms.

Syntactically, the definition of $>_{\mathsf{oh}}$ generalizes that of the first-order $>_{\mathsf{fo}}$. Semantically, the restriction of $>_{\mathsf{oh}}$ to first-order terms coincides with $>_{\mathsf{fo}}$.

The requirements X8 and X10 on $\gg^{\mathsf{f}}$ can be made without loss of generality in a first-order setting.

The quantification over $\mathsf{f} \in \mathit{ghd}(\zeta)$ in rule O3 can be inefficient in an implementation, when different symbols in $\mathit{ghd}(\zeta)$ disagree on which $\gg$ to use. We could generalize the definition of $>_{\mathsf{oh}}$ further to allow underapproximation, but some care would be needed to ensure transitivity. A simple alternative is to enrich all sets $\mathit{ghd}(\zeta)$ that disagree on $\gg$ with a distinguished symbol for which the componentwise extension is used. Since this extension operator is more restrictive than any other ones, whenever it is present in a set $\mathit{ghd}(\zeta)$ there is no need to compute the other ones.

4 Properties

We now state and prove the main properties of our RPO. We focus on the general variant $>_{\mathsf{ho}}$ and show that it is equivalent to the optimized variant $>_{\mathsf{oh}}$ (assuming property X10). Many of the proofs are adapted from Baader and Nipkow [3] and Zantema [38].

Lemma 11. *If $t >_{\mathsf{ho}} s$, then $\mathit{vars}(t) \supseteq \mathit{vars}(s)$.*

As a consequence of Lemma 11, the condition $\mathit{vars}(t) \supseteq \mathit{vars}(\zeta)$ of rule H2 could be written equivalently (but less efficiently) as $\mathit{vars}(t) \supseteq \mathit{vars}(s)$.

Theorem 12 (Transitivity). *If $u >_{\mathsf{ho}} t$ and $t >_{\mathsf{ho}} s$, then $u >_{\mathsf{ho}} s$.*

Proof. By well-founded induction on the multiset $\{|s|, |t|, |u|\}$ with respect to the multiset extension of $>$ on $\mathbb{N}$.

If $u >_{\mathsf{ho}} t$ was derived by rule H1, we have $u = u_1\, u_2$ and $u_k \geq_{\mathsf{ho}} t$ for some k. Since $t >_{\mathsf{ho}} s$ by hypothesis, $u_k >_{\mathsf{ho}} s$ follows either immediately (if $u_k = t$) or by the induction hypothesis (if $u_k >_{\mathsf{ho}} t$). We get $u >_{\mathsf{ho}} s$ by rule H1.

Otherwise, $u >_{\mathsf{ho}} t$ was derived by rule H2 or H3. The *chksubs* condition ensures that u is greater than any immediate subterms of t. We proceed by case analysis on the rule that derived $t >_{\mathsf{ho}} s$.

If $t >_{\mathsf{ho}} s$ was derived by H1, we have $t = t_1\, t_2$ and $t_j \geq_{\mathsf{ho}} s$ for some j. We already noted that $u >_{\mathsf{ho}} t_j$ thanks to $\mathit{chksubs}(u, t)$. In conjunction with $t_j \geq_{\mathsf{ho}} s$, we derive $u >_{\mathsf{ho}} s$ either immediately or by the induction hypothesis.

Otherwise, $t >_{\mathsf{ho}} s$ was derived by rule H2 or H3. The *chksubs* condition ensures that t is greater than any immediate subterms of s. We derive $u >_{\mathsf{ho}} s$ by applying H2 or H3. We first prove $\mathit{chksubs}(u, s)$. The only nontrivial case is $s = s_1\, s_2$. Using $u >_{\mathsf{ho}} t$, we get $u >_{\mathsf{ho}} s_1$ and $u >_{\mathsf{ho}} s_2$ by the induction hypothesis.

If both $u >_{\mathsf{ho}} t$ and $t >_{\mathsf{ho}} s$ were derived by rule H3, we apply H3 to derive $u >_{\mathsf{ho}} s$. This relies on the preservation by $\gg^{\mathsf{f}}_{\mathsf{ho}}$ of transitivity (property X3) on the set consisting of the argument tuples of s, t, u. Transitivity of $>_{\mathsf{ho}}$ on these tuples follows from the induction hypothesis. Finally, if either $u >_{\mathsf{ho}} t$ or $t >_{\mathsf{ho}} s$ was derived by rule H2, we apply H2, relying on the transitivity of $\succ$ and on Lemma 11. □

Theorem 13 (Irreflexivity). $s \not>_{\mathsf{ho}} s$.

Proof. By strong induction on $|s|$. We assume $s >_{\mathsf{ho}} s$ and show that this leads to a contradiction. If $s >_{\mathsf{ho}} s$ was derived by rule H1, we have $s = s_1\, s_2$ with $s_i \geq_{\mathsf{ho}} s$ for some i. Since a term cannot be equal to one of its proper subterms, the comparison is strict. Moreover, we have $s >_{\mathsf{ho}} s_i$ by rule H1. Transitivity yields $s_i >_{\mathsf{ho}} s_i$, contradicting the induction hypothesis. If $s >_{\mathsf{ho}} s$ was derived by rule H2, the contradiction follows immediately from the irreflexivity of $\succ$. Otherwise, $s >_{\mathsf{ho}} s$ was derived by rule H3. Let $s = \zeta\, \bar{s}$. We have $\bar{s} \gg^{\mathsf{f}}_{\mathsf{ho}} \bar{s}$ for all $\mathsf{f} \in \mathit{ghd}(\zeta) \neq \emptyset$. Since $\gg^{\mathsf{f}}$ preserves irreflexivity for transitive relations (property X4) and $>_{\mathsf{ho}}$ is transitive (Theorem 12), there must exist a term $s' \in \bar{s}$ such that $s' >_{\mathsf{ho}} s'$. However, this contradicts the induction hypothesis. □

By Theorems 12 and 13, $>_{\mathsf{ho}}$ is a partial order. In the remaining proofs, we will often leave applications of these theorems (and of antisymmetry) implicit.

Theorem 14 (Subterm Property). *If s is a proper subterm of t, then $t >_{\mathsf{ho}} s$.*

Proof. By structural induction on t, exploiting rule H1 and transitivity of $>_{\mathsf{ho}}$. □

The first-order RPO satisfies compatibility with Σ-operations. A slightly more general property holds for $>_{\mathsf{ho}}$:

Theorem 15 (Compatibility with Functions). *If $t' >_{\mathsf{ho}} t$, then $s\, t'\, \bar{u} >_{\mathsf{ho}} s\, t\, \bar{u}$.*

Proof. By induction on the length of $\bar{u}$. The base case, $\bar{u} = ()$, follows from rule H3, compatibility of $\gg^{\mathsf{f}}$ with tuple contexts (property X6), and the subterm property (Theorem 14). The step case, $\bar{u} = \bar{u}' \cdot u$, also follows from rule H3 and compatibility of $\gg^{\mathsf{f}}$ with contexts. The $\mathit{chksubs}(s\, t'\, \bar{u}'\, u, s\, t\, \bar{u}'\, u)$ condition follows from the induction hypothesis and the subterm property. □

A related property, compatibility with arguments, is useful to rewrite subterms such as $\mathsf{f}\,\mathsf{a}$ in $\mathsf{f}\,\mathsf{a}\,\mathsf{b}$ using a rewrite rule $\mathsf{f}\,x \to t_x$. Unfortunately, $>_{\mathsf{ho}}$ does not enjoy this property: $s' >_{\mathsf{ho}} s$ does not imply $s'\, t >_{\mathsf{ho}} s\, t$. Two counterexamples follow:

1. Given $\mathsf{g} \succ \mathsf{f}$, we have $\mathsf{f}\,\mathsf{g} >_{\mathsf{ho}} \mathsf{g}$ by rule H1, but $\mathsf{f}\,\mathsf{g}\,\mathsf{f} <_{\mathsf{ho}} \mathsf{g}\,\mathsf{f}$ by rule H2.
2. Let $\mathsf{f} \succ \mathsf{b} \succ \mathsf{a}$, and let $\gg^{\mathsf{f}}$ be the lexicographic extension. Then $\mathsf{f}\,\mathsf{a} >_{\mathsf{ho}} \mathsf{f}$ by rule H3, but $\mathsf{f}\,\mathsf{a}\,\mathsf{b} <_{\mathsf{ho}} \mathsf{f}\,\mathsf{b}$ also by rule H3.

The second counterexample and similar ones involving rule H3 can be excluded by requiring that $\gg^{\mathsf{f}}$ is compatible with appending (property X9), which holds for the length-lexicographic and multiset extensions. But there is no way to rule out the first counterexample without losing coincidence with the first-order RPO.

Theorem 16 (Compatibility with Arguments). *Assume that $\gg^{\mathsf{f}}$ is compatible with appending (property X9) for every symbol $\mathsf{f} \in \Sigma$. If $s' >_{\mathsf{ho}} s$ is derivable by rule H2 or H3, then $s'\, t >_{\mathsf{ho}} s\, t$.*

Proof. If $s' >_{\mathsf{ho}} s$ is derivable by rule H2, we apply H2 to derive $s'\,t >_{\mathsf{ho}} s\,t$. To show $\mathit{chksubs}(s'\,t, s\,t)$, we must show that $s'\,t >_{\mathsf{ho}} s$ and $s'\,t >_{\mathsf{ho}} t$. Both are consequences of the subterm property (Theorem 14), together with $s' >_{\mathsf{ho}} s$.

If $s' >_{\mathsf{ho}} s$ is derivable by rule H3, we apply H3 to derive $s'\,t >_{\mathsf{ho}} s\,t$. The condition on the variables of the head of $s'\,t$ can be shown by exploiting the condition on the variables of the head of s'. The *chksubs* condition is shown as above. The condition on the argument tuples follows by property X9. □

Theorem 17 (Stability under Substitution). *If $t >_{\mathsf{ho}} s$, then $t\sigma >_{\mathsf{ho}} s\sigma$ for any substitution σ that respects the mapping ghd.*

Proof. By well-founded induction on the multiset $\{|s|, |t|\}$ with respect to the multiset extension of $>$ on $\mathbb{N}$.

If $t >_{\mathsf{ho}} s$ was derived by rule H1, we have $t = t_1\,t_2$ and $t_j \geq_{\mathsf{ho}} s$ for some j. By the induction hypothesis, $t_j\sigma \geq_{\mathsf{ho}} s\sigma$. Hence, $t\sigma >_{\mathsf{ho}} s\sigma$ by rule H1.

If $t >_{\mathsf{ho}} s$ was derived by rule H2, we have $s = \zeta\,\bar{s}$, $t = \xi\,\bar{t}$, $\xi \succ \zeta$, and $\mathit{chksubs}(t, s)$. We derive $t\sigma >_{\mathsf{ho}} s\sigma$ by applying H2. Since σ respects ghd, we have $\xi\sigma \succ \zeta\sigma$. From $t >_{\mathsf{ho}} s$, we have $\mathit{vars}(t) \supseteq \mathit{vars}(s)$ by Lemma 11 and hence $\mathit{vars}(t\sigma) \supseteq \mathit{vars}(s\sigma) \supseteq \mathit{vars}(\xi\sigma)$. To show $\mathit{chksubs}(t\sigma, s\sigma)$, the nontrivial cases are $s = x$ and is $s = s_1\,s_2$. If $s = x$, then s must be a subterm of t by Lemma 11, and therefore $s\sigma$ is a subterm of $t\sigma$. Thus, we have $t\sigma >_{\mathsf{ho}} s\sigma$ by the subterm property (Theorem 14), from which it is easy to derive $\mathit{chksubs}(t\sigma, s\sigma)$, as desired. If $s = s_1\,s_2$, we get $t >_{\mathsf{ho}} s_1$ and $t >_{\mathsf{ho}} s_2$ from $\mathit{chksubs}(t, s)$. By the induction hypothesis, $t\sigma >_{\mathsf{ho}} s_1\sigma$ and $t >_{\mathsf{ho}} s_2\sigma$, as desired.

If $t >_{\mathsf{ho}} s$ was derived by rule H3, we have $s = \zeta\,\bar{s}$, $t = \zeta\,\bar{t}$, $\bar{t} \gg^{\mathsf{f}}_{\mathsf{ho}} \bar{s}$ for all $\mathsf{f} \in \mathit{ghd}(\zeta)$, and $\mathit{chksubs}(t, s)$. We derive $t\sigma >_{\mathsf{ho}} s\sigma$ by applying H3. Clearly, $s\sigma$ and $t\sigma$ have the same head. The $\mathit{chksubs}(t\sigma, s\sigma)$ condition is proved as for rule H2 above. Finally, we must show that $\bar{t}\sigma \gg^{\mathsf{f}}_{\mathsf{ho}} \bar{s}\sigma$ for all $\mathsf{f} \in \mathit{ghd}(\zeta')$, where $\zeta\sigma = \zeta'\,\bar{u}$ for some $\bar{u}$. Since σ respects ghd, we have $\mathit{ghd}(\zeta') \subseteq \mathit{ghd}(\zeta)$; hence, $\bar{t} \gg^{\mathsf{f}}_{\mathsf{ho}} \bar{s}$ for all $\mathsf{f} \in \mathit{ghd}(\zeta')$. By the induction hypothesis, $t' >_{\mathsf{ho}} s'$ implies $t'\sigma >_{\mathsf{ho}} s'\sigma$ for all $s', t' \in \bar{s} \cup \bar{t}$. By preservation of stability (property X2), we have $\bar{t}\sigma \gg^{\mathsf{f}}_{\mathsf{ho}} \bar{s}\sigma$. By compatibility with prepending (property X8), we get $\bar{u} \cdot \bar{t}\sigma \gg^{\mathsf{f}}_{\mathsf{ho}} \bar{u} \cdot \bar{s}\sigma$, as required to apply H3. □

Theorem 18 (Well-foundedness). *There exists no infinite descending chain $s_0 >_{\mathsf{ho}} s_1 >_{\mathsf{ho}} \cdots$.*

Proof. We assume that there exists a chain $s_0 >_{\mathsf{ho}} s_1 >_{\mathsf{ho}} \cdots$ and show that this leads to a contradiction. If the chain contains nonground terms, we can instantiate all variables by arbitrary terms respecting ghd and exploit stability under substitution (Theorem 17). Thus, we may assume without loss of generality that the terms $s_0, s_1, \ldots$ are ground.

We call a ground term *bad* if it belongs to an infinite descending $>_{\mathsf{ho}}$-chain. Without loss of generality, we assume that s_0 has minimal size among all bad terms and that s_{i+1} has minimal size among all bad terms t such that $s_i >_{\mathsf{ho}} t$.

For each index i, the term s_i must be of the form $\mathsf{f}\,u_1 \ldots u_n$ for some symbol f and ground terms $u_1, \ldots, u_n$. Let $U_i = \emptyset$ if $n = 0$; otherwise, let

$U_i = \{u_1, \ldots, u_n, \mathsf{f}\, u_1 \cdots u_{n-1}\}$. Now let $U = \bigcup_{i=0}^{\infty} U_i$. All terms belonging to U are good: A term from U_0's badness would contradict the minimality of s_0; and if a term $u \in U_{i+1}$ were bad, we would have $s_{i+1} >_{\mathsf{ho}} u$ by rule H1 and $s_i >_{\mathsf{ho}} u$ by transitivity, contradicting the minimality of s_{i+1}.

Next, we show that the only rules that can be used to derive $s_i >_{\mathsf{ho}} s_{i+1}$ are H2 and H3. Suppose H1 were used. Then there would exist a good term $u \in U_i$ such that $u \geq_{\mathsf{ho}} s_{i+1} >_{\mathsf{ho}} s_{i+2}$. This would imply the existence of an infinite chain $u >_{\mathsf{ho}} s_{i+2} >_{\mathsf{ho}} s_{i+3} >_{\mathsf{ho}} \cdots$, contradicting the goodness of u.

Because $\succ$ is well founded and H3 preserves the head symbol, rule H2 can be applied only a finite number of times in the chain. Hence, there must exist an index k such that $s_i >_{\mathsf{ho}} s_{i+1}$ is derived using H3 for all $i \geq k$. Consequently, all terms s_i for $i \geq k$ share the same head symbol f.

Let $s_i = \mathsf{f}\, \bar{u}_i$ for all $i \geq k$. Since H3 is used consistently from index k, we have an infinite $\gg^{\mathsf{f}}_{\mathsf{ho}}$-chain: $\bar{u}_k \gg^{\mathsf{f}}_{\mathsf{ho}} \bar{u}_{k+1} \gg^{\mathsf{f}}_{\mathsf{ho}} \bar{u}_{k+2} \gg^{\mathsf{f}}_{\mathsf{ho}} \cdots$. But since U contains only good terms and comprises all terms occurring in some argument tuple $\bar{u}_i$, $>_{\mathsf{ho}}$ is well founded on U. By preservation of well-foundedness (property X5), $\gg^{\mathsf{f}}_{\mathsf{ho}}$ is well founded. This contradicts the existence of the above $\gg^{\mathsf{f}}_{\mathsf{ho}}$-chain. □

Theorem 19 (Ground Totality). *Assume $\gg^{\mathsf{f}}$ preserves totality (property X7) for every symbol $\mathsf{f} \in \Sigma$, and let s, t be ground terms. Then either $t \geq_{\mathsf{ho}} s$ or $t <_{\mathsf{ho}} s$.*

Proof. By strong induction on $|s| + |t|$. If not $\mathit{chksubs}(t, s)$, then $t \not\succ_{\mathsf{ho}} s_1$ and $t \not\succ_{\mathsf{ho}} s_2$ for $s = s_1\, s_2$. By the induction hypothesis, $s_1 \geq_{\mathsf{ho}} t$ and $s_2 \geq_{\mathsf{ho}} t$. Thus, $s >_{\mathsf{ho}} t$ by rule H1. Analogously, if not $\mathit{chksubs}(s, t)$, then $t >_{\mathsf{ho}} s$. Hence, we may assume $\mathit{chksubs}(t, s)$ and $\mathit{chksubs}(s, t)$. Let $s = \mathsf{f}\, \bar{s}$ and $t = \mathsf{g}\, \bar{t}$. If $\mathsf{g} \succ \mathsf{f}$ or $\mathsf{g} \prec \mathsf{f}$, we have $t >_{\mathsf{ho}} s$ or $s >_{\mathsf{ho}} t$ by rule H2. Otherwise, $\mathsf{f} = \mathsf{g}$. By preservation of totality (property X7), we have either $\bar{t} \gg^{\mathsf{f}}_{\mathsf{ho}} \bar{s}$, $\bar{t} \ll^{\mathsf{f}}_{\mathsf{ho}} \bar{s}$, or $\bar{s} = \bar{t}$. In the first two cases, we have $t >_{\mathsf{ho}} s$ or $t <_{\mathsf{ho}} s$ by rule H3. In the third case, we have $s = t$. □

Having now established the main properties of $>_{\mathsf{ho}}$, we turn to the correspondence between $>_{\mathsf{ho}}$, its optimized variant $>_{\mathsf{oh}}$, and the first-order RPO $>_{\mathsf{fo}}$.

Lemma 20. (1) *If $u >_{\mathsf{oh}} t$ and $t >_{\mathsf{oh}} s$, then $u >_{\mathsf{oh}} s$.* (2) *$s\, t >_{\mathsf{oh}} s$.*

Theorem 21 (Coincidence with Optimized Variant). *Let $>_{\mathsf{ho}}$ and $>_{\mathsf{oh}}$ be orders induced by the same precedence $\succ$ and extension operator family $\gg^{\mathsf{f}}$ (which must satisfy property X10 by the definition of $>_{\mathsf{oh}}$). Then $t >_{\mathsf{ho}} s$ if and only if $t >_{\mathsf{oh}} s$.*

Proof. By strong induction on $|s| + |t|$. The interesting implication is $t >_{\mathsf{ho}} s \Longrightarrow t >_{\mathsf{oh}} s$.

If $t >_{\mathsf{ho}} s$ was derived by rule H1, we have $t = t_1\, t_2$ and $t_j \geq_{\mathsf{ho}} s$ for some j. Hence $t_j \geq_{\mathsf{oh}} s$ by the induction hypothesis, and $t >_{\mathsf{oh}} t_j$ by Lemma 20(2) or rule O1. We get $t >_{\mathsf{oh}} s$ either immediately or by Lemma 20(1).

If $t >_{\mathsf{ho}} s$ was derived by rule H2, we derive $t >_{\mathsf{oh}} s$ by applying O2. We must show that $\mathit{chksubs}$ implies $\mathit{chkargs}$. We have $s = s_1\, s_2$ with $t >_{\mathsf{ho}} s_1$ and $t >_{\mathsf{ho}} s_2$. Let $s = \zeta\, \bar{s}\, s_2$. We must show that $t >_{\mathsf{oh}} s'$ for all $s' \in \bar{s} \cup \{s_2\}$. If $s' = s_2$,

we have $t >_{\mathsf{ho}} s_2$ immediately. Otherwise, from $t >_{\mathsf{ho}} s_1$, we have $t >_{\mathsf{ho}} s'$ by the subterm property (Theorem 14). In both cases, we get $\mathit{chkargs}(t, \bar{s})$ by the induction hypothesis.

If $t >_{\mathsf{ho}} s$ was derived by rule H3, we derive $t >_{\mathsf{oh}} s$ by applying O3. The $\mathit{chkargs}(t, \bar{s})$ condition is proved as in the H2 case. From $\bar{t} \gg^{\mathsf{f}}_{\mathsf{ho}} \bar{s}$, we derive $\bar{t} \gg^{\mathsf{f}}_{\mathsf{oh}} \bar{s}$ by the induction hypothesis and monotonicity of $\gg^{\mathsf{f}}$ (property X1). □

Corollary 22 (Coincidence with First-Order RPO). *Let* $>_{\mathsf{ho}}$ *and* $>_{\mathsf{fo}}$ *be orders induced by the same precedence* $\succ$ *and extension operator family* $\gg^{\mathsf{f}}$ *satisfying minimality of the empty tuple (property X10). Then* $>_{\mathsf{ho}}$ *and* $>_{\mathsf{fo}}$ *coincide on first-order terms.*

5 Examples

Although our motivation was to design a term order suitable for higher-order superposition, we can use $>_{\mathsf{ho}}$ (and $>_{\mathsf{oh}}$) to show the termination of λ-free higher-order term rewriting systems or, equivalently, applicative term rewriting systems [24]. We present a selection of examples of how this can be done, illustrating the strengths and weaknesses of the order in this context. Many of the examples are taken from the literature. Since $>_{\mathsf{ho}}$ coincides with the standard RPO on first-order terms, we consider only examples featuring higher-order constructs.

To establish termination of a term rewriting system, a standard approach is to show that all of its rewrite rules $t \to s$ can be oriented as $t > s$ by a single *reduction order*: a well-founded partial order that is compatible with contexts and stable under substitutions. Regrettably, $>_{\mathsf{ho}}$ is not a reduction order since it lacks compatibility with arguments. But the conditional Theorem 16 is often sufficient in practice. Assuming that the extension operator is compatible with appending (property X9), we may apply H2 and H3 to orient rewrite rules. Moreover, we may even use H1 for rewrite rules that operate on non-function terms; supplying an argument to a non-function would violate typing. To identify non-functions and to restrict instantiations, we assume that terms respect the typing discipline of the simply typed λ-calculus. Together, property X9 and the restriction on the application of H1 achieve the same effect as η-saturation [19].

For simplicity, the examples are all monolithic, but a modern termination prover would use the dependency pair framework [2] to break down a large term rewriting system into smaller components that can be analyzed separately. Unless mentioned otherwise, the RPO instances considered employ the length-lexicographic extension operator. We consistently use italics for variables and sans serif for symbols

Example 23. Consider the following term rewriting system:

$$\mathsf{insert}\ (f\ n)\ (\mathsf{image}\ f\ A) \xrightarrow{1} \mathsf{image}\ f\ (\mathsf{insert}\ n\ A) \qquad \mathsf{square}\ n \xrightarrow{2} \mathsf{times}\ n\ n$$

Rule 1 captures a set-theoretic property: $\{f(n)\} \cup f[A] = f[\{n\} \cup A]$. We can prove termination using $>_{\mathsf{ho}}$: By letting $\mathsf{insert} \succ \mathsf{image}$ and $\mathsf{square} \succ \mathsf{times}$,

both rules can be oriented by H2. In contrast, rule 2 is beyond the reach of the applicative order $>_{\mathsf{ap}}$ for the same reason that $\mathsf{g}\ x \not>_{\mathsf{ap}} \mathsf{f}\ x\ x$ in Example 9. The system is also beyond the scope of the uncurrying approach of Hirokawa et al. [19] because of the variable application $f\ n$.

Example 24. The following system specifies a map function on an ML-style option type equipped with two constructors, None and Some:

$$\mathsf{omap}\ f\ \mathsf{None} \xrightarrow{1} \mathsf{None} \qquad \mathsf{omap}\ f\ (\mathsf{Some}\ n) \xrightarrow{2} \mathsf{Some}\ (f\ n)$$

To establish termination, it would appear that it suffices to apply H2 to orient both rules, using a precedence such that $\mathsf{omap} \succ \mathsf{None}, \mathsf{Some}$. However, a closer inspection reveals that the *chksubs* condition blocks the application of H2 to orient rule 2: We would need $\mathsf{omap}\ f\ (\mathsf{Some}\ n) >_{\mathsf{ho}} f\ n$, which cannot be established without further assumptions. With a typing discipline that distinguishes between options and other data, f cannot be instantiated by a term having omap as its head. Thus, we can safely restrict $\mathit{ghd}(f)$ to $\Sigma - \{\mathsf{omap}\}$ and assign the highest precedence to omap. We then have $\mathsf{omap}\ f\ (\mathsf{Some}\ n) >_{\mathsf{ho}} f\ n$ by H2, as required to orient rule 2.

The above example suggests a general strategy for coping with variables that occur unapplied on the left-hand side of a rewrite rule and applied on the right-hand side.

Example 25. The next system is taken from Lysne and Piris [28, Example 5], with an additional rule adapted from Lifantsev and Bachmair [27, Example 6]:

$$\mathsf{iter}\ f\ n\ \mathsf{Nil} \xrightarrow{1} n \qquad \mathsf{sum}\ ms \xrightarrow{3} \mathsf{iter}\ \mathsf{plus}\ 0\ ms$$
$$\mathsf{iter}\ f\ n\ (\mathsf{Cons}\ m\ ms) \xrightarrow{2} \mathsf{iter}\ f\ (f\ n\ m)\ ms \qquad \mathsf{iter}\ \mathsf{times}\ 1\ ms \xrightarrow{4} \mathsf{prod}\ ms$$

The iter function is a general iterator on lists of numbers. Reasoning about the types, we can safely take $\mathit{ghd}(f) = \Sigma - \{\mathsf{iter}, \mathsf{sum}\}$. By letting $\mathsf{sum} \succ \mathsf{iter}$ and ensuring that iter is greater than any other symbol, rule 1 can be oriented by H1, rule 2 can be oriented by H3, and rules 3 and 4 can be oriented by H2. The application of H1 is legitimate if numbers are distinguished from functions.

Example 26. The following rules are taken from Jouannaud and Rubio [22, Sect. 4.2]:

$$\mathsf{fmap}\ x\ \mathsf{Nil} \to \mathsf{Nil} \qquad \mathsf{fmap}\ x\ (\mathsf{Cons}\ f\ fs) \to \mathsf{Cons}\ (f\ x)\ (\mathsf{fmap}\ x\ fs)$$

The fmap function applies each function from a list to a value x and returns the list of results. The typing discipline allows us to take $\mathit{ghd}(f) = \Sigma - \{\mathsf{fmap}\}$. By making fmap greater than any other symbol, both rules can be oriented by H2.

Example 27. The next system is from Toyama [35, Example 4]:

$$\mathsf{ite}\ \mathsf{true}\ xs\ ys \xrightarrow{1} xs \qquad \mathsf{filter}\ q\ \mathsf{Nil} \xrightarrow{3} \mathsf{Nil}$$
$$\mathsf{ite}\ \mathsf{false}\ xs\ ys \xrightarrow{2} ys \qquad \mathsf{filter}\ q\ (\mathsf{Cons}\ x\ xs) \xrightarrow{4} \mathsf{ite}\ (q\ x)\ (\mathsf{Cons}\ x\ (\mathsf{filter}\ q\ xs))\ (\mathsf{filter}\ q\ xs)$$

The typing discipline allows us to take $\mathit{ghd}(q) = \Sigma - \{\mathsf{filter}\}$. Given $\mathsf{filter} \succ \mathsf{f}$ for all $\mathsf{f} \in \Sigma$, rules 1 and 2 can be oriented by H1, and rules 3 and 4 can be oriented by H2. The application of H1 is legitimate if lists are distinguished from functions.

Example 28. Sternagel and Thiemann [32, Example 1] compare different approaches to uncurrying on the following system:

$$\begin{aligned} \mathsf{minus}\,\mathsf{0} &\xrightarrow{1} \mathsf{K}\,\mathsf{0} & \qquad \mathsf{K}\,m\,n &\xrightarrow{5} m \\ \mathsf{minus}\,m\,\mathsf{0} &\xrightarrow{2} m & \qquad \mathsf{map}\,f\,\mathsf{Nil} &\xrightarrow{6} \mathsf{Nil} \\ \mathsf{minus}\,m\,m &\xrightarrow{3} \mathsf{0} & \qquad \mathsf{map}\,f\,(\mathsf{Cons}\,m\,ms) &\xrightarrow{7} \mathsf{Cons}\,(f\,m)\,(\mathsf{map}\,f\,ms) \\ \mathsf{minus}\,(\mathsf{S}\,m)\,(\mathsf{S}\,n) &\xrightarrow{4} \mathsf{minus}\,m\,n \end{aligned}$$

The minus function implements subtraction on Peano numbers, whereas map applies a function elementwise to a finite list. We establish termination by employing $>_{\mathsf{ho}}$ with a precedence such that $\mathsf{minus} \succ \mathsf{K}, \mathsf{0}$ and $\mathsf{map} \succ \mathsf{Cons}$. Rules 2, 5, and 6 are oriented by H1; rules 1, 3, and 7 are oriented by H2; and rule 4 is oriented by H3. The application of H1 is legitimate if numbers and lists are distinguished from functions.

Example 29. Lifantsev and Bachmair [27, Example 8] define a higher-order function that applies its first argument twice to its second argument: $\mathsf{twice}\,f\,x \to f\,(f\,x)$. This rewrite rule is problematic in our framework, because we cannot rely on the typing discipline to prevent the instantiation of f by a term with twice as its head. Indeed, $\mathsf{twice}\,(\mathsf{twice}\,\mathsf{S})$ is a natural way to specify the function $x \mapsto \mathsf{S}\,(\mathsf{S}\,(\mathsf{S}\,(\mathsf{S}\,x)))$.

Example 30. Toyama's recursor specification [35, Example 6] exhibits the same limitation in a more general context:

$$\mathsf{rec}\,n\,f\,\mathsf{0} \to n \qquad\qquad \mathsf{rec}\,n\,f\,(\mathsf{S}\,m) \to f\,(\mathsf{S}\,m)\,(\mathsf{rec}\,n\,f\,m)$$

Example 31. Let $\mathit{ghd}(f) = \mathit{ghd}(g) = \{\mathsf{prod}\}$, and consider the system

$$\mathsf{plus}\,\mathsf{0}\,m \xrightarrow{1} m \quad \mathsf{plus}\,(\mathsf{S}\,m)\,n \xrightarrow{2} \mathsf{plus}\,m\,(\mathsf{S}\,n) \quad f\,\mathsf{prod} \xrightarrow{3} f \quad f\,(g\,m) \xrightarrow{4} f\,m\,g$$

These rules can be used to simplify nested prod terms; for example: $\mathsf{prod}\,(\mathsf{prod}\,\mathsf{a}\,\mathsf{b}) \xrightarrow{4} \mathsf{prod}\,\mathsf{b}\,(\mathsf{prod}\,\mathsf{a}) \xrightarrow{4} \mathsf{prod}\,\mathsf{b}\,\mathsf{a}\,\mathsf{prod} \xrightarrow{3} \mathsf{prod}\,\mathsf{b}\,\mathsf{a}$. The $>_{\mathsf{ho}}$ order can be employed by taking $\gg^{\mathsf{prod}}$ to be the multiset extension and by relying on typing to orient rule 1 with H1. The applicative order $>_{\mathsf{ap}}$ fails because a combination of lexicographic and multiset extensions is needed to orient rules 2 and 4. The uncurrying approach of Hirokawa et al. [19] also fails because of the applied variables on the left-hand side of rule 4.

Carsten Fuhs, a developer of the AProVE termination prover, generously offered to apply his tool to our examples, expressed as untyped applicative term rewriting systems. Using AProVE's web interface with a 60 s time limit, he could establish the termination of Examples 23, 24, 28, 29, and 31. The tool timed out for Examples 25–27 and 30. For Example 31, the tool found a complex proof

involving several applications of linear polynomial interpretations, dependency pairs, and 2×2 matrix interpretations (to cope with rule 4). Although our focus is on superposition, it would be interesting to implement the new RPO in a tool such as AProVE and to conduct a more systematic evaluation on standard higher-order termination benchmarks against higher-order termination provers such as THOR [12] and WANDA [25].

6 Discussion

Rewriting of λ-free higher-order terms has been amply studied in the literature, under various names such as applicative term rewriting [24] and simply typed term rewriting [37]. Translations from higher-order to first-order term rewriting systems were designed by Aoto and Yamada [1], Toyama [35], Hirokawa et al. [19], and others. Toyama also studied S-expressions, a formalism that regards $((\mathsf{f}\,\mathsf{a})\,\mathsf{b})$ and $(\mathsf{f}\,\mathsf{a}\,\mathsf{b})$ as distinct. For higher-order terms with λ-abstraction, various frameworks have been proposed, including Nipkow's higher-order rewrite systems [30], Blanqui's inductive data type systems [9], and Kop's algebraic functional systems with metavariables [25]. Kop's thesis [25, Chapter 3] includes a comprehensive overview.

When designing our RPO $>_{\mathsf{ho}}$, we aimed at full coincidence with the first-order case. Our goal is to gradually transform first-order automatic provers into higher-order provers. By carefully generalizing the proof calculi and data structures, we aim at designing provers that behave like first-order provers on first-order problems, perform mostly like first-order provers on higher-order problems that are mostly first-order, and scale up to arbitrary higher-order problems.

The simplicity of $>_{\mathsf{ho}}$ fails to do justice to the labor of exploring the design space. Methodologically, the use of a proof assistant [31] equipped with a model finder [6] and automatic theorem provers [5] was invaluable for designing the orders, proving their properties, and carrying out various experiments. As one example among many, at a late stage in the design process, we generalized the rules H2 and O2 to allow variable heads. Thanks to the tool support, which keeps track of what must be changed, it took us less than one hour to adapt the main proofs and convince ourselves that the new approach worked, and a few more hours to complete the proofs. Performing such changes on paper is a less reliable, and less satisfying, enterprise. Another role of the formal proofs is to serve as companions to the informal proofs, clarifying finer points. Term rewriting lends itself well to formalization in proof assistants, perhaps because it requires little sophisticated mathematics beyond well-founded induction and recursion. The CoLoR library by Blanqui and Koprowski [11], in Coq, the C*i*ME3 toolkit by Contejean et al. [15], also in Coq, and the IsaFoR library by Thiemann and Sternagel [34], in Isabelle/HOL, have already explored this territory, providing formalized metatheory but also certified termination and confluence checkers.

The $>_{\mathsf{ho}}$ order is in some ways less flexible than the hybrid curried–uncurried approaches, where the currying is one more parameter that can be adjusted. In exchange, it raises the level of abstraction, by providing a uniform view of

higher-order terms, and it works in the open-world setting of higher-order proof search. For example, consider the proof obligation $\exists g.\ \forall x, y.\ g\ x\ y = \mathsf{f}\ y\ x$ and the SK combinator definitions $\forall x, y.\ \mathsf{K}\ x\ y = x$ and $\forall x, y, z.\ \mathsf{S}\ x\ y\ z = x\ z\ (y\ z)$. A prover will need to synthesize the witness $\mathsf{S}\ (\mathsf{K}\ (\mathsf{S}\ \mathsf{f}))\ \mathsf{K}$, representing $\lambda x\ y.\ \mathsf{f}\ y\ x$, for the existential variable g. A hybrid approach such as HORPO might infer arity 2 for f based on the problem, but then the witness, in which f appears unapplied, cannot be expressed.

An open question is whether it is possible to design an order that largely coincides with the first-order RPO while enjoying compatibility with arbitrary contexts. This could presumably be achieved by weakening rule H1 and strengthening the *chksubs* condition of H2 and H3 accordingly; so far, our attempts have resulted only in a rediscovery of the applicative RPO.

For superposition, richer type systems would be desirable. These could be incorporated either by simply ignoring the types, by encoding them in the terms, or by generalizing the order. Support for λ-abstraction would be useful but challenging. Any well-founded order enjoying the subterm property would need to distinguish β-equivalent terms, to exclude the cycle $\mathsf{a} =_\beta (\lambda x.\ \mathsf{a})\ (\mathsf{f}\ \mathsf{a}) > \mathsf{f}\ \mathsf{a} > \mathsf{a}$. We could aim at compatibility with β-reduction, but even this property might be irrelevant for higher-order superposition. It might even be preferable to avoid λ-abstractions altogether, by relying on SK combinators or by adding new symbols and their definitions during proof search.

Acknowledgment. We are grateful to Stephan Merz, Tobias Nipkow, and Christoph Weidenbach for making this research possible; to Heiko Becker and Dmitriy Traytel for proving some theorems on the lexicographic and multiset extensions in Isabelle/HOL; to Carsten Fuhs for his invaluable feedback and his experiments; to Alexander Steen for pointing us to related work; and to Simon Cruanes, Mark Summerfield, Dmitriy Traytel, and the anonymous reviewers for suggesting textual improvements.

Blanchette has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 713999, Matryoshka). Wand is supported by the Deutsche Forschungsgemeinschaft (DFG) grant Hardening the Hammer (NI 491/14-1).

References

1. Aoto, T., Yamada, T.: Termination of simply typed term rewriting by translation and labelling. In: Nieuwenhuis, R. (ed.) RTA 2003. LNCS, vol. 2706, pp. 380–394. Springer, Heidelberg (2003). doi:10.1007/3-540-44881-0_27
2. Arts, T., Giesl, J.: Termination of term rewriting using dependency pairs. Theor. Comput. Sci. **236**(1–2), 133–178 (2000)
3. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press, Cambridge (1998)
4. Benzmüller, C., Miller, D.: Automation of higher-order logic. In: Siekmann, J.H. (ed.) Computational Logic, Handbook of the History of Logic, vol. 9, pp. 215–254. Elsevier (2014)
5. Blanchette, J.C., Kaliszyk, C., Paulson, L.C., Urban, J.: Hammering towards QED. J. Formalized Reasoning **9**(1), 101–148 (2016)

6. Blanchette, J.C., Nipkow, T.: Nitpick: A counterexample generator for higher-order logic based on a relational model finder. In: Kaufmann, M., Paulson, L.C. (eds.) ITP 2010. LNCS, vol. 6172, pp. 131–146. Springer, Heidelberg (2010). doi:10.1007/978-3-642-14052-5_11
7. Blanchette, J.C., Waldmann, U., Wand, D.: Formalization of recursive path orders for lambda-free higher-order terms. Archive of Formal Proofs, formal proof development (2016). https://isa-afp.org/entries/Lambda_Free_RPOs.shtml
8. Blanchette, J.C., Waldmann, U., Wand, D.: A lambda-free higher-order recursive path order. Technical report (2016). http://people.mpi-inf.mpg.de/jblanche/lambda_free_rpo_rep.pdf
9. Blanqui, F.: Termination and confluence of higher-order rewrite systems. In: Bachmair, L. (ed.) RTA 2000. LNCS, vol. 1833, pp. 47–61. Springer, Heidelberg (2000). doi:10.1007/10721975_4
10. Blanqui, F., Jouannaud, J., Rubio, A.: The computability path ordering. Log. Meth. Comput. Sci. 11(4) (2015)
11. Blanqui, F., Koprowski, A.: CoLoR: A Coq library on well-founded rewrite relations and its application to the automated verification of termination certificates. Math. Struct. Comput. Sci. **21**(4), 827–859 (2011)
12. Bofill, M., Borralleras, C., Rodríguez-Carbonell, E., Rubio, A.: The recursive path and polynomial ordering for first-order and higher-order terms. J. Log. Comput. **23**(1), 263–305 (2013)
13. Bofill, M., Rubio, A.: Paramodulation with non-monotonic orderings and simplification. J. Autom. Reasoning **50**(1), 51–98 (2013)
14. Codish, M., Giesl, J., Schneider-Kamp, P., Thiemann, R.: SAT solving for termination proofs with recursive path orders and dependency pairs. J. Autom. Reasoning **49**(1), 53–93 (2012)
15. Contejean, É., Courtieu, P., Forest, J., Pons, O., Urbain, X.: Automated certified proofs with CiME3. In: Schmidt-Schauß, M. (ed.) Rewriting Techniques and Applications (RTA 2011). Leibniz International Proceedings in Informatics (LIPIcs), vol. 10, pp. 21–30. Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2011)
16. Dershowitz, N., Manna, Z.: Proving termination with multiset orderings. Commun. ACM **22**(8), 465–476 (1979)
17. Ganzinger, H., Hagen, G., Nieuwenhuis, R., Oliveras, A., Tinelli, C.: DPLL(T): Fast decision procedures. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 175–188. Springer, Heidelberg (2004). doi:10.1007/978-3-540-27813-9_14
18. Henkin, L.: Completeness in the theory of types. J. Symb. Log. **15**(2), 81–91 (1950)
19. Hirokawa, N., Middeldorp, A., Zankl, H.: Uncurrying for termination and complexity. J. Autom. Reasoning **50**(3), 279–315 (2013)
20. Huet, G., Oppen, D.C.: Equations and rewrite rules: A survey. In: Book, R.V. (ed.) Formal Language Theory: Perspectives and Open Problems, pp. 349–405. Academic Press (1980)
21. Hughes, R.J.M.: Super-combinators: A new implementation method for applicative languages. In: ACM Symposium on LISP and Functional Programming (LFP 1982), pp. 1–10. ACM Press (1982)
22. Jouannaud, J.-P., Rubio, A.: A recursive path ordering for higher-order terms in η-long β-normal form. In: Ganzinger, H. (ed.) RTA 1996. LNCS, vol. 1103, pp. 108–122. Springer, Heidelberg (1996). doi:10.1007/3-540-61464-8_46
23. Jouannaud, J., Rubio, A.: Polymorphic higher-order recursive path orderings. J. ACM **54**(1), 2:1–2:48 (2007)
24. Kennaway, R., Klop, J.W., Sleep, M.R., de Vries, F.: Comparing curried and uncurried rewriting. J. Symb. Comput. **21**(1), 15–39 (1996)

25. Kop, C.: Higher Order Termination. Ph.D. thesis, Vrije Universiteit Amsterdam (2012)
26. Kop, C., Raamsdonk, F.: A higher-order iterative path ordering. In: Cervesato, I., Veith, H., Voronkov, A. (eds.) LPAR 2008. LNCS (LNAI), vol. 5330, pp. 697–711. Springer, Heidelberg (2008). doi:10.1007/978-3-540-89439-1_48
27. Lifantsev, M., Bachmair, L.: An LPO-based termination ordering for higher-order terms without λ-abstraction. In: Grundy, J., Newey, M. (eds.) TPHOLs 1998. LNCS, vol. 1479, pp. 277–293. Springer, Heidelberg (1998). doi:10.1007/BFb0055142
28. Lysne, O., Piris, J.: A termination ordering for higher order rewrite systems. In: Hsiang, J. (ed.) RTA 1995. LNCS, vol. 914, pp. 26–40. Springer, Heidelberg (1995). doi:10.1007/3-540-59200-8_45
29. Nieuwenhuis, R., Rubio, A.: Paramodulation-based theorem proving. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. I, pp. 371–443. Elsevier and MIT Press (2001)
30. Nipkow, T.: Higher-order critical pairs. In: Logic in Computer Science (LICS 1991), pp. 342–349. IEEE Computer Society (1991)
31. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic. LNCS, vol. 2283. Springer, Heidelberg (2002)
32. Sternagel, C., Thiemann, R.: Generalized and formalized uncurrying. In: Tinelli, C., Sofronie-Stokkermans, V. (eds.) FroCoS 2011. LNCS (LNAI), vol. 6989, pp. 243–258. Springer, Heidelberg (2011). doi:10.1007/978-3-642-24364-6_17
33. Thiemann, R., Allais, G., Nagele, J.: On the formalization of termination techniques based on multiset orderings. In: Tiwari, A. (ed.) Rewriting Techniques and Applications (RTA 2012). LIPIcs, vol. 15, pp. 339–354. Schloss Dagstuhl–Leibniz-Zentrum für Informatik (2012)
34. Thiemann, R., Sternagel, C.: Certification of termination proofs using CeTA. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs 2009. LNCS, vol. 5674, pp. 452–468. Springer, Heidelberg (2009). doi:10.1007/978-3-642-03359-9_31
35. Toyama, Y.: Termination of S-expression rewriting systems: lexicographic path ordering for higher-order terms. In: Oostrom, V. (ed.) RTA 2004. LNCS, vol. 3091, pp. 40–54. Springer, Heidelberg (2004). doi:10.1007/978-3-540-25979-4_3
36. Turner, D.A.: A new implementation technique for applicative languages. Softw.: Pract. Experience **9**(1), 31–49 (1979)
37. Yamada, T.: Confluence and termination of simply typed term rewriting systems. In: Middeldorp, A. (ed.) RTA 2001. LNCS, vol. 2051, pp. 338–352. Springer, Heidelberg (2001). doi:10.1007/3-540-45127-7_25
38. Zantema, H.: Termination. In: Bezem, M., Klop, J.W., de Vrijer, R. (eds.) Term Rewriting Systems. Cambridge Tracts in Theoretical Computer Science, vol. 55, pp. 181–259. Cambridge University Press (2003)

Automated Constructivization of Proofs

Frédéric Gilbert(✉)

École des Ponts ParisTech, Inria, CEA LIST, Marne-la-Vallée, France
frederic.a.gilbert@inria.fr

Abstract. No computable function can output a constructive proof from a classical one whenever its associated theorem also holds constructively. We show in this paper that it is however possible, in practice, to turn a large amount of classical proofs into constructive ones. We describe for this purpose a linear-time constructivization algorithm which is provably complete on large fragments of predicate logic.

1 Introduction

Classical and constructive provability match on several specific sets of propositions. In propositional logic, as a consequence of Glivenko's theorem [1], a formula $\neg A$ is a classical theorem iff it is a constructive one. In arithmetic, a Π_2^0 proposition is a theorem in Peano arithmetic iff it is a theorem in Heyting arithmetic [2].

We present in this paper an efficient constructivization algorithm Construct for predicate logic in general, from cut-free classical sequent calculus **LK** to constructive sequent calculus **LJ**. Unlike the two previous examples, constructivization in predicate logic is as hard as constructive theorem proving. Therefore, as we expect Construct to terminate, Construct is incomplete in the sense that it may terminate with a failure output.

Construct consists of three **linear-time** steps:

1. An algorithm Normalize, designed to push occurrences of the right weakening rule towards the root in **LK** proofs. Its purpose is to limit the number of propositions appearing at the right-hand side of sequents in **LK** proofs.
2. A partial translation from cut-free **LK** to a new constructive system **LI**. This algorithm is referred to as Annotate as the **LI** system is designed as **LK** equipped with specific annotations – making it a constructive system. Annotate is the only step which may fail.
3. A complete translation Interpret from **LI** to **LJ**.

The Normalize step taken alone leads to a simple yet efficient constructivization algorithm Weak construct, which is defined to succeed whenever the result of Normalize happens to be directly interpretable in **LJ**, i.e. to have at most one proposition on the right-hand side of sequents in its proof.

The main property of Construct is to be provably **complete** on large fragments of predicate logic, in the sense that for any proposition A in one

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 480–495, 2017.
DOI: 10.1007/978-3-662-54458-7_28

of these fragments, CONSTRUCT is ensured to terminate successfully on any cut-free **LK** proof of A. Such fragments for which classical and constructive provability match will be referred to as **constructive fragments**. For instance, as a consequence of Glivenko's theorem [1], the set of negated propositions is a **constructive fragment** of propositional logic. The completeness properties of CONSTRUCT lead to the following results:

- The identification of a new constructive fragment F, the fragment of assertions containing no negative occurrence of the connective $\vee$ and no positive occurrence of the connective $\Rightarrow$. Both WEAK CONSTRUCT and CONSTRUCT are provably complete on F.
- The completeness of CONSTRUCT on two already known constructive fragments. The first one, referred to as F_{Ku}, appears as the set of fix points of a polarized version of Kuroda's double-negation translation [3,4]. The second one, referred to as F_{Ma}, appears as a set of assertions for which any cut-free **LK** proof can be directly interpreted as a proof in Maehara's multi-succedent calculus [5]. Hence, the completeness of CONSTRUCT on these two fragments yields a uniform proof of two results coming from very different works.

After the introduction of basic notations and definitions, the two already known constructive fragments F_{Ku} and F_{Ma} are presented. Then, the NORMALIZE step is presented along with the simple constructivization algorithm WEAK CONSTRUCT. In the following section, the new constructive fragment F is defined, and WEAK CONSTRUCT is proved complete on F. Then, the full constructivization algorithm CONSTRUCT is introduced together with the proof of its completeness on F, F_{Ku} and F_{Ma}. In the last part, experimental results of constructivization using WEAK CONSTRUCT and CONSTRUCT are presented. These experiments are based the classical theorem prover **Zenon** [10] and the constructive proof checker **Dedukti** [9].

2 Notations and Definitions

In the following, we only consider as primitive the connectives and quantifiers $\forall$, $\exists$, $\wedge$, $\vee$, $\Rightarrow$ and $\bot$. $\neg A$ is defined as $A \Rightarrow \bot$. $\top$, which doesn't appear in this paper, could be defined as $\bot \Rightarrow \bot$.

We use a definition of sequents based on **multisets**. The size of a multiset Γ will be referred to as $|\Gamma|$. We will use the notation (A) to refer to a multiset containing either zero or one element. Given a multiset $\Gamma = A_1, \cdots, A_n$, we will use the notations $\neg\Gamma$ and $\Gamma \Rightarrow B$ as shorthands for $\neg A_1, \cdots, \neg A_n$, and $A_1 \Rightarrow B, \cdots, A_n \Rightarrow B$ respectively. Finally, we use the notation $\bigvee$ to refer to an arbitrary encoding of the n-ary disjunction from the binary one – using $\bot$ for the nullary case.

Definition 1. *We define the cut-free classical sequent calculus **LK** with the following rules:*

$$\frac{}{\bot \vdash}\ \bot_L \qquad \frac{}{A \vdash A}\ axiom$$

$$\frac{\Gamma \vdash \Delta}{\Gamma, \Gamma' \vdash \Delta}\ weak_L \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, \Delta'}\ weak_R$$

$$\frac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta}\ contr_L \qquad \frac{\Gamma \vdash A, A, \Delta}{\Gamma \vdash A, \Delta}\ contr_R$$

$$\frac{\Gamma, A, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\ \wedge_L \qquad \frac{\Gamma \vdash A, \Delta \qquad \Gamma \vdash B, \Delta}{\Gamma \vdash A \wedge B, \Delta}\ \wedge_R$$

$$\frac{\Gamma, A \vdash \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta}\ \vee_L \qquad \frac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash A \vee B, \Delta}\ \vee_R$$

$$\frac{\Gamma \vdash A, \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \Rightarrow B \vdash \Delta}\ \Rightarrow_L \qquad \frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \Rightarrow B, \Delta}\ \Rightarrow_R$$

$$\frac{\Gamma, A[t/x] \vdash \Delta}{\Gamma, \forall x A \vdash \Delta}\ \forall_L \qquad \frac{\Gamma \vdash A, \Delta}{\Gamma \vdash \forall x A, \Delta}\ \forall_R$$

$$\frac{\Gamma, A \vdash \Delta}{\Gamma, \exists x A \vdash \Delta}\ \exists_L \qquad \frac{\Gamma \vdash A[t/x], \Delta}{\Gamma \vdash \exists x A, \Delta}\ \exists_R$$

with the standard freshness constraints for the variables introduced in the rules $\forall_R$ *and* $\exists_L$*.*

Definition 2. *We define the constructive sequent calculus* ***LJ*** *from* ***LK****, applying the following changes:*

- *All rules except* $contr_R$, $\vee_R$, $\Rightarrow_L$ *are restricted to sequents with at most one proposition on the right-hand side of sequents.*

  For instance, $\wedge_R$ *becomes* $\dfrac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ \wedge_R$

- *There is no* $contr_R$ *rule*

- *The* $\vee_R$ *rule is split into two rules* $\dfrac{\Gamma \vdash A_i}{\Gamma \vdash A_0 \vee A_1}\ \vee_R$

- *The* $\Rightarrow_L$ *rule becomes* $\dfrac{\Gamma \vdash A \qquad \Gamma, B \vdash (C)}{\Gamma, A \Rightarrow B \vdash (C)}\ \Rightarrow_L$

– *We add a cut rule* $\dfrac{\Gamma \vdash A \qquad \Gamma, A \vdash (B)}{\Gamma \vdash (B)}\, cut$

Remark 1. In these presentations of **LK** and **LJ**,

– weakenings are applied to multisets instead of propositions

– $\bot_L$ and *axiom* are not relaxed to $\dfrac{}{\Gamma, \bot \vdash \Delta}\,\bot_L$ and $\dfrac{}{\Gamma, A \vdash A, \Delta}\, axiom$

These specific conventions are chosen to ease the definition of the algorithm Normalize in Sect. 5, which requires pushing weakenings towards the root of the proof.

Definition 3. *We introduce the following notations in* ***LK****, along with their constructive analogs in* ***LJ****:*

– $\dfrac{}{\Gamma, A \vdash A, \Delta}\, axiom^*$ *for* $\dfrac{\dfrac{\dfrac{}{A \vdash A}\, axiom}{\Gamma, A \vdash A}\, weak_L}{\Gamma, A \vdash A, \Delta}\, weak_R$

– $\dfrac{}{\Gamma, \bot \vdash \Delta}\,\bot_L^*$ *for* $\dfrac{\dfrac{\dfrac{}{\bot \vdash}\,\bot_L}{\Gamma, \bot \vdash}\, weak_L}{\Gamma, \bot \vdash \Delta}\, weak_R$

– $\dfrac{\Gamma \vdash A, \Delta}{\Gamma, \neg A \vdash \Delta}\,\neg_L$ *for* $\dfrac{\Gamma \vdash A, \Delta \qquad \dfrac{}{\Gamma, \bot \vdash \Delta}\,\bot_L^*}{\Gamma, \neg A \vdash \Delta}\,\Rightarrow_L$

– $\dfrac{\Gamma, A \vdash \Delta}{\Gamma \vdash \neg A, \Delta}\,\neg_R$ *for* $\dfrac{\dfrac{\Gamma, A \vdash \Delta}{\Gamma, A \vdash \bot, \Delta}\, weak_R}{\Gamma \vdash \neg A, \Delta}\,\Rightarrow_R$

3 State of the Art: Two Constructive Fragments of Predicate Logic

Constructive sequent calculus – as well as constructive natural deduction – extends the notion of constructive provability from propositions to sequents of the shape $\Gamma \vdash (G)$, which will be referred to as **mono-succedent sequents**. As a consequence, we will define constructive fragments of predicate logic as sets of mono-succedent sequents instead of sets of simple propositions.

The definitions of these fragments will be based on the usual notion of polarity of occurrences of connectives, quantifiers and atoms in a sequent: given a sequent $\Gamma \vdash \Delta$,

– the root of a proposition in Γ is negative, the root of a proposition in Δ is positive

– polarity only changes between an occurrence of $A \Rightarrow B$ and the occurrence of its direct subformula A (in particular, as $\neg A$ is defined as $A \Rightarrow \bot$, it changes between $\neg A$ and its direct subformula A).

Definition 4. *We define the following fragments of predicate logic:*

– F_{Ku}*, the fragment of sequents of the shape* $\Gamma \vdash$ *containing no positive occurrence of* $\forall$*.*
– F_{Ma}*, the fragment of mono-succedent sequents containing no positive occurrence of* $\forall$ *and no positive occurrence of* $\Rightarrow$*.*

Theorem 1. F_{Ku} *is a constructive fragment of predicate logic: for any sequent* $\Gamma \vdash$ *in* F_{Ku}*,* $\Gamma \vdash$ *is classically provable iff it is constructively provable.*

The key arguments to prove this theorem as an adaptation of Kuroda's double negation translation [3] are the following:

1. Kuroda's double negation translation [3] is based on a double negation translation $|\cdot|_{Ku}$ inserting double-negations after any occurrence of $\forall$. The original theorem is that a proposition A is classically provable iff $\neg\neg|A|_{Ku}$ is constructively provable.
2. It can adapted in two ways. First, $|\cdot|_{Ku}$ can be lightened to insert double negations only after positive occurrences of $\forall$ as shown in [4], and extended from propositions to contexts. Second, the main statement can be turned to the following one: a classical sequent $\Gamma \vdash \Delta$ is classically provable iff $|\Gamma, \neg\Delta|_{Ku} \vdash$ is constructively provable.
3. By definition of F_{Ku}, a sequent $\Gamma \vdash$ in F_{Ku} admits the property $\Gamma = |\Gamma|_{Ku}$, hence $\Gamma \vdash$ is classically provable iff it is constructively provable.

We don't give more details on this proof as the completeness of Construct on F_{Ku} shown in Sect. 6 will yield a new proof of this result.

Remark 2. One could expect similar constructive fragments to be found using other double negation translations, such as Gödel-Gentzen's [6,7] or Kolmogorov's [8]. Unfortunately, these two translations always insert double-negations in front of atoms, hence they cannot be easily modified to leave a large fragment of propositions unchanged.

Theorem 2. F_{Ma} *is a constructive fragment of predicate logic: for any sequent* $\Gamma \vdash (G)$ *in* F_{Ma}*,* $\Gamma \vdash (G)$ *is classically provable iff it is constructively provable.*

It lays on a key idea: polarity restrictions have a direct influence on the shape of cut-free proofs. It can be presented in the following way:

Lemma 1. *For any connective or quantifier* X *and any cut-free* ***LK*** *proof* Π *of a sequent* $\Gamma \vdash \Delta$*:*

– *If* $\Gamma \vdash \Delta$ *contains no positive occurrence of* X*, then* Π *doesn't contain the rule* X_R*.*

– *If* $\Gamma \vdash \Delta$ *contains no negative occurrence of* X*, then* Π *doesn't contain the rule* X_L*.*

This lemma can be proved directly by induction on cut-free **LK** proofs. Using this lemma, the key arguments to prove Theorem 2 are the following:

1. All **LK** rules except $\Rightarrow_R$ and $\forall_R$ rules belong in Maehara's multi-succedent calculus [5], a constructive multi-succedent sequent calculus.
2. By Lemma 1, F_{Ma} sequents are proved by cut-free **LK** proofs without the $\Rightarrow_R$ and $\forall_R$ rules.
3. Hence, a sequent $\Gamma \vdash (G)$ in F_{Ma} is classically provable iff it is constructively provable.

Again, we don't give more details on this proof as the completeness of Construct on F_{Ma} shown in Sect. 6 will yield a new proof of this result.

Remark 3. The same fragment F_{Ma} can be found using similar multi-succedent constructive systems, such as Dragalin's calculus GHPC [11].

4 The Weakening Normalization

A naive constructivization algorithm can be defined by selecting **LK** proofs which can be directly interpreted in **LJ**.

In this direct interpretation, premises of the classical rules $\vee_R$ and $\Rightarrow_L$ may be multi-succedent only when they are introduced by a $weak_R$ whose premise is a mono-succedent sequent. For instance, the classical derivation

$$\dfrac{\dfrac{\Gamma \vdash A}{\Gamma \vdash A, B}\, weak_R}{\Gamma \vdash A \vee B}\, \vee_R \text{ can be interpreted as } \dfrac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\, \vee_R \,.$$

However, in practice, the $weak_R$ rule doesn't appear as low as possible – in presentations using multi-succedents *axiom* rules, they may not appear at all. Such situations are problematic for constructive interpretations: for instance, a classical proof such as

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\, axiom}{A \vdash A, B}\, weak_R}{\vdash A \Rightarrow A, B}\, \Rightarrow_R}{\vdash (A \Rightarrow A) \vee B}\, \vee_R$$

cannot be interpreted in **LJ** directly because the $weak_R$ rule doesn't occur immediately above the $\vee_R$ rule.

The Normalize algorithm is designed to address this issue, pushing the application of $weak_R$ as low as possible in proofs. In its definition, we need to consider all possible configuration of $weak_R$ appearing above a **LK** rule. In order to factor this definition, we partition all such configurations into three classes **A**, **B**, and **C**.

These definitions will be based on the following notation of **LK** proofs:

Definition 5. *We write any cut-free* ***LK*** *rule X as*

$$\frac{\Gamma, L_1 \vdash R_1, \Delta \qquad \cdots \qquad \Gamma, L_n \vdash R_n, \Delta}{\Gamma, L \vdash R, \Delta}\, X$$

where $L_1, \cdots, L_n$, $R_1, \cdots, R_n$, L and R are the (possibly empty) ***multisets*** *of propositions containing the active propositions of the rule X.*

For instance, in the rule $\dfrac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \Rightarrow B, \Delta} \Rightarrow_R$,

$L_1 = \{A\}$, $R_1 = \{B\}$, $L = \emptyset$, and $R = \{A \Rightarrow B\}$.

The classes **A**, **B**, and **C** are defined as follows:

Definition 6. *We consider all configurations where $weak_R$ appears above a* ***LK*** *rule X, in its i-th premise:*

$$\frac{\cdots \qquad \dfrac{\Gamma, L_i \vdash \Delta_i}{\Gamma, L_i \vdash R_i, \Delta}\, weak_R \qquad \cdots}{\Gamma, L \vdash R, \Delta}\, X$$

This weakening can be done on propositions in R_i, in Δ or both: in the general case, we only know $\Delta_i \subseteq (R_i, \Delta)$. We define the following partition of all cases:

- ***A****: $R_i \subseteq \Delta_i$*
- ***B****: $R_i \not\subseteq \Delta_i$ and $\Delta_i \subseteq \Delta$*
- ***C****: $R_i \not\subseteq \Delta_i$ and $\Delta_i \not\subseteq \Delta$. This only happens when $|R_i| = 2$, when exactly one proposition of R_i is in Δ_i.*

Definition 7. NORMALIZE *is a linear-time algorithm associating any cut-free* ***LK*** *proof of a sequent $\Gamma \vdash \Delta$ to a proof of a sequent $\Gamma \vdash \Delta'$, where $\Delta' \subseteq \Delta$. It is defined recursively. Using the conventions of Definition 5, we describe the original proof Π as*

$$\frac{\dfrac{\Pi_1}{\Gamma, L_1 \vdash R_1, \Delta} \qquad \cdots \qquad \dfrac{\Pi_n}{\Gamma, L_n \vdash R_n, \Delta}}{\Gamma, L \vdash R, \Delta}\, X$$

The definition of NORMALIZE*(Π) is based on the analysis of the proof*

$$\frac{\dfrac{\text{NORMALIZE}(\Pi_1)}{\Gamma, L_1 \vdash \Delta_1}\, weak_R \qquad \cdots \qquad \dfrac{\text{NORMALIZE}(\Pi_n)}{\Gamma, L_n \vdash \Delta_n}\, weak_R}{\Gamma, L \vdash R, \Delta}\, X$$

The different cases are the following:

– *Case **1**: for all index i, **A** holds, i.e. $R_i \subseteq \Delta_i$.*

If X is $weak_R$, we define NORMALIZE*(Π) as* NORMALIZE*(Π_1).*

Else, writing $\Delta_i = R_i, \Delta'_i$, we define NORMALIZE*($\Pi$) as*

$$\dfrac{\dfrac{\text{NORMALIZE}(\Pi_1)}{\dfrac{\Gamma, L_1 \vdash R_1, \Delta'_1}{\Gamma, L_1 \vdash R_1, \Delta'}\, weak_R} \quad \cdots \quad \dfrac{\dfrac{\text{NORMALIZE}(\Pi_n)}{\Gamma, L_n \vdash R_n, \Delta'_n}}{\Gamma, L_n \vdash R_n, \Delta'}\, weak_R}{\Gamma, L \vdash R, \Delta'}\, X$$

where Δ' is the smallest multiset containing all multisets Δ'_i

– *Case **2**: there exists a smallest premise i for which **B** holds, i.e. $R_i \not\subseteq \Delta_i$ and $\Delta_i \subseteq \Delta$. As $R_i \neq \emptyset$, ether X is $\Rightarrow_R$ or $L_i = \emptyset$.*

If X is $\Rightarrow_R$, we define NORMALIZE*(Π) as* $\dfrac{\dfrac{\text{NORMALIZE}(\Pi_1)}{\dfrac{\Gamma, A \vdash \Delta_1}{\Gamma, A \vdash B, \Delta_1}\, weak_R}}{\Gamma \vdash A \Rightarrow B, \Delta_1}\, \Rightarrow_R$

Else, $L_i = \emptyset$ and we define NORMALIZE*(Π) as* $\dfrac{\dfrac{\text{NORMALIZE}(\Pi_i)}{\Gamma \vdash \Delta_i}}{\Gamma, L \vdash \Delta_i}\, weak_L$

– *Case **3**: there exists a smallest premise i for which the case **C** applies, i.e. $R_i \not\subseteq \Delta_i$ and $\Delta_i \not\subseteq \Delta$. This only happens when $|R_i| = 2$, when exactly one proposition of R_i is in Δ_i. In this case, X is either $contr_R$ or $\vee_R$.*

If X is $contr_R$, we can write $R_1 = A, A$, and $\Delta_1 = (A, \Delta'_1)$ with $\Delta'_1 \subseteq \Delta$. We define NORMALIZE*($\Pi$) as* NORMALIZE*($\Pi_1$).*

If X is $\vee_R$, we can write $R_1 = A_0, A_1$, and $\Delta_1 = (A_k, \Delta'_1)$ with $\Delta'_1 \subseteq \Delta$.

We define NORMALIZE*(Π) as* $\dfrac{\dfrac{\text{NORMALIZE}(\Pi_1)}{\Gamma \vdash A_k, \Delta'_1}\, weak_R}{\Gamma \vdash A_0 \vee A_1, \Delta'_1}\, \vee_R$

Remark 4. The nullary rules *axiom* and $\bot_L$ having no premise, they match the first case.

Definition 8. *We define a first constructivization algorithm* WEAK CONSTRUCT*, which*

– *takes as input a cut-free **LK** proof* $\dfrac{\Pi}{\Gamma \vdash (G)}$,

– *computes the proof* $\dfrac{\text{NORMALIZE}(\Pi)}{\Gamma \vdash (G)}\, weak_R$,

– *outputs its **LJ** interpretation if it exists and fails otherwise.*

5 A New Constructive Fragment

Definition 9. *We define F as the fragment of mono-succedent sequents containing no negative occurrence of $\vee$ and no positive occurrence of $\Rightarrow$.*

Theorem 3. WEAK CONSTRUCT *is complete on F: if Π is a cut-free **LK** proof of a sequent $\Gamma \vdash (G) \in F$, then* WEAK CONSTRUCT(Π) *succeeds.*

Proof. By Lemma 1, F sequents are proved by cut-free **LK** proofs containing no $\vee_L$ or $\Rightarrow_R$ rule. We prove that for any such proof Π, NORMALIZE(Π) proves a mono-succedent sequent interpretable in **LJ**. This proof is done by induction on cut-free **LK** proofs containing no $\vee_L$ or $\Rightarrow_R$ rule, following the partition of cases and the notations introduced in the definition of NORMALIZE:

– Case **1**: we split this case according to the rule X.

- nullary rules: *axiom* and $\bot_L$ are interpretable in **LJ**.
- $weak_R$: The result follows directly by induction hypothesis.
- other unary rules: In these cases $\Delta' = \Delta'_1$, hence NORMALIZE(Π) is

$$\dfrac{\dfrac{\dfrac{\text{NORMALIZE}(\Pi_1)}{\Gamma, L_1 \vdash R_1, \Delta'_1}\, weak_R}{\Gamma, L_1 \vdash R_1, \Delta'_1}\, X}{\Gamma, L \vdash R, \Delta'_1}$$

  By induction hypothesis, NORMALIZE(Π_1) is interpretable in **LJ**. Hence, $|R_1| \leq 1$, which ensures that X is neither $contr_R$ nor $\vee_R$. All other unary rules lead to a proof interpretable in **LJ**, therefore the result is interpretable in **LJ**.

- $\vee_L$: This case doesn't occur by hypothesis

- $\Rightarrow_L$: By induction hypothesis, NORMALIZE(Π_1) and NORMALIZE(Π_2) are interpretable in **LJ**, hence $|R_1, \Delta'_1| \leq 1$. As $|R_1| = 1$, $\Delta'_1 = \emptyset$, and $\Delta' = \Delta'_2$.

  As $\dfrac{\dfrac{\Gamma \vdash A}{\Gamma \vdash A, \Delta'_2}\, weak_R \qquad \dfrac{\Gamma, B \vdash \Delta'_2}{\Gamma, B \vdash \Delta'_2}\, weak_R}{\Gamma, A \Rightarrow B \vdash \Delta'_2} \Rightarrow_L$

  is interpretable as $\dfrac{\Gamma \vdash A \qquad \Gamma, B \vdash \Delta'_2}{\Gamma, A \Rightarrow B \vdash \Delta'_2} \Rightarrow_L$ in **LJ**, the result follows.

- $\wedge_R$: By induction hypothesis, NORMALIZE(Π_1) and NORMALIZE(Π_2) are interpretable in **LJ**, hence $|R_1, \Delta'_1| \leq 1$ and $|R_2, \Delta'_2| \leq 1$. As $|R_1| = |R_2| = 1$, $\Delta'_1 = \Delta'_2 = \emptyset$. Therefore $\Delta' = \emptyset$, from which the result follows.

– Case **2**: By hypothesis, X is not $\Rightarrow_R$, hence NORMALIZE(Π) is defined as

$$\dfrac{\dfrac{\textsc{Normalize}(\Pi_i)}{\Gamma \vdash \Delta_i}}{\Gamma, L \vdash \Delta_i}\, weak_L$$

The result follows by induction hypothesis.

– Case **3**: If X is $contr_R$, the result follows directly by induction hypothesis. Else, X is $\vee_R$. By induction hypothesis, $\textsc{Normalize}(\Pi_1)$ is interpretable in **LJ**, thus $|A_k, \Delta_1'| \leq 1$, and $\Delta_1' = \emptyset$.

As $\dfrac{\dfrac{\Gamma \vdash A_k}{\Gamma \vdash A_0, A_1}\, weak_R}{\Gamma \vdash A_0 \vee A_1}\, \vee_R$ is interpretable as $\dfrac{\Gamma \vdash A_k}{\Gamma \vdash A_0 \vee A_1}\, \vee_R$ in **LJ**, the result follows.

Corollary 1. *The fragment F is a constructive fragment of predicate logic: a sequent $\Gamma \vdash (G)$ is classically provable iff it is constructively provable.*

6 The Full Constructivization Algorithm

The previous algorithm Weak construct was based on the reject of multi-succedent sequents. The idea leading to our main algorithm Construct is to try to interpret multi-succedent sequents constructively as well. This interpretation is based on a new multi-succedent constructive system, which will be referred to as **LI** in the following. As mentioned in the introduction, the constructivization algorithm Construct comprises three steps: first the algorithm Normalize, then a partial translation Annotate from **LK** proofs to **LI** proofs, and finally a complete translation Interpret from **LI** proof to **LJ** proofs.

There are several ways to interpret multi-succedent sequents constructively. For instance, $\Gamma \vdash \bigvee \Delta$ and $\Gamma, \neg\Delta \vdash$ are two possible interpretations of a multi-succedent sequent $\Gamma \vdash \Delta$. These interpretation are equivalent classically but not constructively: for instance, the classical sequent $\vdash A, \neg A$ is not provable constructively under the first interpretation, but it is provable constructively under the second one. As a consequence, some classical rules may be constructively valid or not according to the chosen interpretation of classical sequents.

The new system **LI** is built to benefit from the freedom left in the constructive interpretation of classical sequents. **LI** is designed as a sequent calculus based on **annotated sequents**, where the annotation will refer to the choice of constructive interpretation of the underlying classical sequent. We formalize first the notion of **annotated sequents**.

Definition 10. *We define the set of **annotated sequents** as sequents of the shape $\Gamma \vdash \Delta_1; \Delta_2$.*

We define the following interpretation Interpret *on annotated sequents:* $\textsc{Interpret}(\Gamma \vdash \Delta_1; \Delta_2) \quad = \quad \Gamma, \neg\Delta_2 \vdash \bigvee \Delta_1$.

*In the following, this function will be extended from **LI** proofs to **LJ** proofs.*

We define the following erasure function Erase *on annotated sequents:*

$\textsc{Erase}(\Gamma \vdash \Delta_1; \Delta_2) \quad = \quad \Gamma \vdash \Delta_1, \Delta_2.$

*In the following, this function will be extended from **LI** proofs to **LK** proofs.*

Then, we define the system **LI** in the following way:

Definition 11. ***LI*** *is based on the following rules:*

$$\frac{}{\bot \vdash;}\ \bot_L \qquad \frac{}{A \vdash A;}\ axiom^1 \qquad \frac{}{A \vdash; A}\ axiom^2$$

$$\frac{\Gamma \vdash \Delta_1; \Delta_2}{\Gamma, \Gamma' \vdash \Delta_1; \Delta_2}\ weak_L \qquad \frac{\Gamma \vdash \Delta_1; \Delta_2}{\Gamma \vdash \Delta_1, \Delta_1'; \Delta_2, \Delta_2'}\ weak_R$$

$$\frac{\Gamma, A, A \vdash \Delta_1; \Delta_2}{\Gamma, A \vdash \Delta_1; \Delta_2}\ contr_L \qquad \frac{\Gamma \vdash A, A, \Delta_1; \Delta_2}{\Gamma \vdash A, \Delta_1; \Delta_2}\ contr_R^1 \qquad \frac{\Gamma \vdash \Delta_1; A, A, \Delta_2}{\Gamma \vdash \Delta_1; A, \Delta_2}\ contr_R^2$$

$$\frac{\Gamma, A, B \vdash \Delta_1; \Delta_2}{\Gamma, A \wedge B \vdash \Delta_1; \Delta_2}\ \wedge_L \qquad \frac{\Gamma \vdash A, \Delta_1; \Delta_2 \qquad \Gamma \vdash B, \Delta_1; \Delta_2}{\Gamma \vdash A \wedge B, \Delta_1; \Delta_2}\ \wedge_R^1$$

$$\frac{\Gamma \vdash; A, \Delta_2 \qquad \Gamma \vdash; B, \Delta_2}{\Gamma \vdash; A \wedge B, \Delta_2}\ \wedge_R^2 \qquad \frac{\Gamma \vdash A, \Delta_1; \Delta_2 \qquad \Gamma \vdash B, \Delta_1; \Delta_2}{\Gamma \vdash \Delta_1; A \wedge B, \Delta_2}\ \wedge_R^3,\ |\Delta_1| \geq 1$$

$$\frac{\Gamma, A \vdash \Delta_1; \Delta_2 \qquad \Gamma, B \vdash \Delta_1; \Delta_2}{\Gamma, A \vee B \vdash \Delta_1; \Delta_2}\ \vee_L$$

$$\frac{\Gamma \vdash A, B, \Delta_1; \Delta_2}{\Gamma \vdash A \vee B, \Delta_1; \Delta_2}\ \vee_R^1 \qquad \frac{\Gamma \vdash \Delta_1; A, B, \Delta_2}{\Gamma \vdash \Delta_1; A \vee B, \Delta_2}\ \vee_R^2$$

$$\frac{\Gamma \vdash; A, \Delta_2 \qquad \Gamma, B \vdash; \Delta_2}{\Gamma, A \Rightarrow B \vdash; \Delta_2}\ \Rightarrow_L^1 \qquad \frac{\Gamma \vdash A, \Delta_1; \Delta_2 \qquad \Gamma, B \vdash \Delta_1; \Delta_2}{\Gamma, A \Rightarrow B \vdash \Delta_1; \Delta_2}\ \Rightarrow_L^2,\ |\Delta_1| \geq 1$$

$$\frac{\Gamma, A \vdash B; \Delta_2}{\Gamma \vdash A \Rightarrow B; \Delta_2}\ \Rightarrow_R^1 \qquad \frac{\Gamma, A \vdash; B, \Delta_2}{\Gamma \vdash; A \Rightarrow B, \Delta_2}\ \Rightarrow_R^2$$

$$\frac{\Gamma, A[t/x] \vdash \Delta_1; \Delta_2}{\Gamma, \forall x A \vdash \Delta_1; \Delta_2}\ \forall_L \qquad \frac{\Gamma \vdash A; \Delta_2}{\Gamma \vdash \forall x A; \Delta_2}\ \forall_R^1 \qquad \frac{\Gamma \vdash A; \Delta_2}{\Gamma \vdash; \forall x A, \Delta_2}\ \forall_R^2$$

$$\frac{\Gamma, A \vdash \Delta_1; \Delta_2}{\Gamma, \exists x A \vdash \Delta_1; \Delta_2}\ \exists_L \qquad \frac{\Gamma \vdash A[t/x], \Delta_1; \Delta_2}{\Gamma \vdash \exists x A, \Delta_1; \Delta_2}\ \exists_R^1 \qquad \frac{\Gamma \vdash \Delta_1; A[t/x], \Delta_2}{\Gamma \vdash \Delta_1; \exists x A, \Delta_2}\ \exists_R^2$$

with the standard freshness constraints for the variables introduced in the rules $\forall_R^i$ and $\exists_L$.

All **LI** rules correspond to a **LK** rule through the erasure of the premises and the conclusions. Hence, we can extend the Erase function from **LI** rules to **LK** rules, and consequently from **LI** proofs to **LK** proofs.

In the same way, we would like to extend the Interpret function from **LI** proofs to **LJ** proofs. This can done associating each **LI** rule to a partial **LJ** proof deriving the interpretation of its conclusion from the interpretation of its premises. However, such an approach would be heavy: as the disjunction in **LJ** is binary, $\bigvee$ is a based on a nesting of binary disjunctions, and a proposition in $\Gamma \vdash \Delta_1; \Delta_2$ can occur deep in $\Gamma, \neg\Delta_2 \vdash \bigvee \Delta_1$. As Interpret will be part of

the constructivization algorithm CONSTRUCT, we need to find another method to define it as a linear-time algorithm.

For this reason, we will define the interpretation of rules using the property that $\Gamma \vdash \bigvee \Delta$ is constructively provable iff $\Gamma, \Delta \Rightarrow G \vdash G$ is provable for any proposition G.

Definition 12. *We define the function* $\text{INTERPRET}'(\cdot|G)$ *on annotated sequents as* $\text{INTERPRET}'(\Gamma \vdash \Delta_1; \Delta_2|G) = (\Gamma, \Delta_1 \Rightarrow G, \neg\Delta_2 \vdash G)$.

We extend $\text{INTERPRET}'$ *from* ***LI*** *rules to partial* ***LJ*** *derivations in the following way:*

From a ***LI*** *rule* $\dfrac{\Gamma^1 \vdash \Delta_1^1; \Delta_2^1 \qquad \cdots \qquad \Gamma^n \vdash \Delta_1^n; \Delta_2^n}{\Gamma \vdash \Delta_1; \Delta_2} R$

and a proposition G*, we define a partial* ***LJ*** *derivation* $\text{INTERPRET}'(R|G)$ *as a partial derivation of the form*

$$\dfrac{\dfrac{\text{INTERPRET}'(\Gamma^1 \vdash \Delta_1^1; \Delta_2^1|G^1) \qquad \cdots \qquad \text{INTERPRET}'(\Gamma^n \vdash \Delta_1^n; \Delta_2^n|G^n)}{\vdots}}{\text{INTERPRET}'(\Gamma \vdash \Delta_1; \Delta_2|G)}$$

The ***LI*** *system is designed to ensure that such definitions rely on simple constructive tautologies. As an illustration, we present here the case of the rule*

$$\dfrac{\Gamma \vdash A, \Delta_1; \Delta_2 \qquad \Gamma, B \vdash \Delta_1; \Delta_2}{\Gamma, A \Rightarrow B \vdash \Delta_1; \Delta_2} \Rightarrow_L^3$$

From a proposition G*, defining* $\Sigma = \Gamma, \Delta_1 \Rightarrow G, \neg\Delta_2$*, we derive*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Sigma, A \vdash A}\, axiom^* \qquad \dfrac{\Sigma, B \vdash G}{\Sigma, B, A \vdash G}\, weak_L}{\Sigma, A \Rightarrow B, A \vdash G} \Rightarrow_L}{\Sigma, A \Rightarrow B \vdash A \Rightarrow G} \Rightarrow_R \qquad \dfrac{\Sigma, A \Rightarrow G \vdash G}{\Sigma, A \Rightarrow B, A \Rightarrow G \vdash G}\, weak_L}{\Sigma, A \Rightarrow B \vdash G}\, cut$$

where the two open premises correspond to $\text{INTERPRET}'(\Gamma, B \vdash \Delta_1; \Delta_2|G)$ *and* $\text{INTERPRET}'(\Gamma \vdash A, \Delta_1; \Delta_2|G)$ *respectively.*

Remark 5. In this case, we chose $G_1 = G_2 = G$*. Other choices for* G_i *appear in the cases* $\wedge_R^2$*,* $\Rightarrow_L^1$*,* $\Rightarrow_R^2$*, and* $\forall_R^2$*.*

In a second step, we extend $\text{INTERPRET}'(\cdot|G)$ *from* ***LI*** *proofs to* ***LJ*** *proofs recursively. Finally, we extend* $\text{INTERPRET}(\cdot)$ *from* ***LI*** *proofs of sequents of the shape* $\Gamma \vdash (G)$*; to* ***LJ*** *proofs:*

– *for* Π *a* ***LI*** *proof of a sequent* $\Gamma \vdash;$*, we define* $\text{INTERPRET}(\Pi)$ *as*

$$\dfrac{\dfrac{}{\Gamma, \bot \vdash}\, \bot_L^* \qquad \dfrac{\text{INTERPRET}'(\Pi|\bot)}{\Gamma \vdash \bot}}{\Gamma \vdash}\, cut$$

– *for Π a **LI** proof of a sequent $\Gamma \vdash G$;, we define* $\textsc{Interpret}(\Pi)$ *as*

$$\dfrac{\dfrac{\textsc{Interpret}'(\Pi|G)}{\Gamma, G \Rightarrow G \vdash G} \qquad \dfrac{\dfrac{}{\Gamma, G \vdash G}\, axiom^*}{\Gamma \vdash G \Rightarrow G} \Rightarrow_R}{\Gamma \vdash G}\, cut$$

Definition 13. *We define the linear-time partial algorithm* $\textsc{Annotate}(\cdot|\cdot)$ *with, as inputs, a **LI** sequent S and a cut-free **LK** proof Π of* $\textsc{Erase}(S)$ *and, as output, either a **LI** proof of S or a failure. This annotation is done from the root to the leaves: at each step, the first argument S prescribe how the current conclusion must be annotated. The definition is recursive on the second argument.*

Describing S as $\Gamma \vdash \Delta_1; \Delta_2$ and Π as $\dfrac{\dfrac{\Pi^1}{\Gamma^1 \vdash \Delta^1} \quad \cdots \quad \dfrac{\Pi^n}{\Gamma^n \vdash \Delta^n}}{\Gamma \vdash \Delta_1, \Delta_2} R$,

– *If there exists a **LI** rule*

$$\dfrac{\Gamma^1 \vdash \Delta_1^1; \Delta_2^1 \qquad \cdots \qquad \Gamma^n \vdash \Delta_1^n; \Delta_2^n}{\Gamma \vdash \Delta_1; \Delta_2}\, R'$$

such that for all i, $\Delta_1^i, \Delta_2^i = \Delta^i$, then the output is

$$\dfrac{\dfrac{\textsc{Annotate}(\Gamma^1 \vdash \Delta_1^1; \Delta_2^1|\Pi^1)}{\Gamma^1 \vdash \Delta_1^1; \Delta_2^1} \quad \cdots \quad \dfrac{\textsc{Annotate}(\Gamma^n \vdash \Delta_1^n; \Delta_2^n|\Pi^n)}{\Gamma^n \vdash \Delta_1^n; \Delta_2^n}}{\Gamma \vdash \Delta_1; \Delta_2}\, R'$$

– *Else,* $\textsc{Annotate}(\cdot, \cdot)$ *fails.*

Remark 6. The only failing cases appear when the rule R is either $\Rightarrow_R$ or $\forall_R$, and exclusively for sequents $\Gamma \vdash \Delta_1; \Delta_2$ such that $|\Delta_1, \Delta_2| > 1$.

Definition 14. *We define the linear-time constructivization algorithm* Construct, *which*

– *takes as input a cut-free **LK** proof Π of a sequent $\Gamma \vdash (G)$,*
– *computes the proof* $\Pi' = \dfrac{\textsc{Normalize}(\Pi)}{\Gamma \vdash (G)}\, weak_R$,
– *outputs* $\textsc{Interpret}(\textsc{Annotate}(\Gamma \vdash (G); |\Pi'))$ *if it exists and fails otherwise.*

Example 1. We consider the law of excluded middle $A \vee \neg A$ given with the following **LK** proof: $\dfrac{\dfrac{\dfrac{}{A \vdash A}\, axiom}{\vdash A, \neg A} \Rightarrow_R}{\vdash A \vee \neg A} \vee_R$. This proof is unchanged by Normalize.

The Annotate step fails as follows: $\dfrac{\dfrac{\textsc{Failure}}{\vdash A, \neg A;}}{\vdash A \vee \neg A;}\, \vee_R^1$

Example 2. We consider a variant of the non contradiction of law of excluded middle, $(\neg(A \vee \neg A)) \Rightarrow B$, given with the proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A, B}\,axiom^*}{\vdash A, \neg A, B}\Rightarrow_R}{\vdash A \vee \neg A, B}\vee_R \qquad \dfrac{}{\bot \vdash B}\,\bot_L^*}{\neg(A \vee \neg A) \vdash B}\Rightarrow_L}{\vdash (\neg(A \vee \neg A)) \Rightarrow B}\Rightarrow_R$$

The result of NORMALIZE is

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\,axiom}{\vdash A, \neg A}\Rightarrow_R}{\vdash A \vee \neg A}\vee_R \qquad \dfrac{}{\bot \vdash}\,\bot_L}{\neg(A \vee \neg A) \vdash}\Rightarrow_L}{\neg(A \vee \neg A) \vdash B}\,weak_R}{\vdash (\neg(A \vee \neg A)) \Rightarrow B}\Rightarrow_R$$

Then, the result of ANNOTATE is

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash; A}\,axiom^2}{\vdash; A, \neg A}\Rightarrow^2_R}{\vdash; A \vee \neg A}\vee^2_R \qquad \dfrac{}{\bot \vdash;}\,\bot_L}{\neg(A \vee \neg A) \vdash;}\Rightarrow^1_L}{\neg(A \vee \neg A) \vdash B;}\,weak_R}{\vdash (\neg(A \vee \neg A)) \Rightarrow B;}\Rightarrow^1_R$$

As ANNOTATE is the only step which may fail, CONSTRUCT succeeds on this example. We see on the example that the application of NORMALIZE was crucial for ANNOTATE to succeed.

Theorem 4. CONSTRUCT *is complete on* F, F_{Ku}, *and* F_{Ma}: *for any proof* Π *of a sequent* S *in one of these fragments,* CONSTRUCT(Π) *succeeds.*

Proof. We consider F, F_{Ku}, and F_{Ma} separately:

– For F: we consider a cut-free **LK** proof Π of a sequent $\Gamma \vdash (G) \in F$.

By Theorem 3, $\Pi' = \dfrac{\text{NORMALIZE}(\Pi)}{\Gamma \vdash (G)}\,weak_R$ is interpretable in **LJ**.

As a consequence, the only multi-succedent sequents in Π' are conclusions of weakenings. As all failing cases (c.f. Remark 6) involve sequents $\Gamma \vdash \Delta_1; \Delta_2$ such that $|\Delta_1, \Delta_2| > 1$ which are conclusions of $\Rightarrow_R$ or $\forall_R$ rules, ANNOTATE succeeds. Hence, CONSTRUCT succeeds.

– For F_{Ku}: the result follows directly from a stronger assertion: for any cut-free **LK** proof Π of a sequent $\Gamma \vdash \Delta$ containing no $\forall_R$ rule, ANNOTATE$(\Gamma \vdash; \Delta | \Pi)$ succeeds. This assertion is proved by induction on such sequents and proofs, noticing that all induction hypotheses refer to sequents of the shape $\Gamma' \vdash; \Delta'$.
– For F_{Ma}: we consider a cut-free **LK** proof Π of a sequent in F_{Ma}. As mentioned in Remark 6 the only failing cases involve the $\Rightarrow_R$ or $\forall_R$ rules, which don't occur in a proof of a sequent in F_{Ma}. Hence, CONSTRUCT succeeds.

7 Experimental Results

In order to measure the success of Construct in practice, experiments were made on the basis of **TPTP** [13] first-order problems. The classical theorem prover **Zenon** [10] was used to prove such problems. **Zenon** builds cut-free **LK** proofs internally. It was instrumented to use these internal proofs as inputs for an implementation of Weak construct and Construct. The **LJ** proofs obtained as outputs were expressed and checked in the constructive logical framework **Dedukti** [9].

A set of 724 **TPTP** problems was selected for the experimentations, corresponding to all **TPTP** problems in the category **FOF** which could be proved in less than 1 s using the uninstrumented version of **Zenon**. The results are the following:

- Weak construct led to constructive proofs in 51% of tested cases.
- Construct led to constructive proofs in 85% of tested cases (including all Weak construct successes).

All constructive proofs generated were successfully checked using **Dedukti**. Among all cases where Construct failed, 35% are proved to be unvalid constructively using the constructive theorem prover **ileanCoP** [12].

References

1. Glivenko, V.: Sur quelques points de la logique de M. Brouwer. Bulletins de la classe des sciences **15**(5), 183–188 (1929)
2. Friedman, H.: Classically and intuitionistically provably recursive functions. In: Müller, G.H., Scott, D.S. (eds.) Higher Set Theory. LNM, vol. 669, pp. 21–27. Springer, Heidelberg (1978). doi:10.1007/BFb0103100
3. Kuroda, S., et al.: Intuitionistische untersuchungen der formalistischen logik. Nagoya Math. J. **2**, 35–47 (1951)
4. Boudard, M., Hermant, O.: Polarizing double-negation translations. In: McMillan, K., Middeldorp, A., Voronkov, A. (eds.) LPAR 2013. LNCS, vol. 8312, pp. 182–197. Springer, Heidelberg (2013). doi:10.1007/978-3-642-45221-5_14
5. Maehara, S., et al.: Eine darstellung der intuitionistischen logik in der klassischen. Nagoya Math. J. **7**, 45–64 (1954)
6. Gentzen, G.: Über das verhältnis zwischen intuitionistischer und klassischer arithmetik. Arch. Math. Logic **16**(3), 119–132 (1974)
7. Gödel, K.: Zur intuitionistischen arithmetik und zahlentheorie. Ergebnisse eines mathematischen Kolloquiums **4**(1933), 34–38 (1933)
8. Kolmogorov, A.N.: On the principle of excluded middle. Mat. Sb **32**(646–667), 24 (1925)
9. Boespflug, M., Carbonneaux, Q., Hermant, O.: The $\lambda\Pi$-calculus modulo as a universal proof language. In: Pichardie, D., Weber, T. (eds.) PxTP (2012)
10. Bonichon, R., Delahaye, D., Doligez, D.: Zenon: an extensible automated theorem prover producing checkable proofs. In: Dershowitz, N., Voronkov, A. (eds.) LPAR 2007. LNCS (LNAI), vol. 4790, pp. 151–165. Springer, Heidelberg (2007). doi:10.1007/978-3-540-75560-9_13

11. Dragalin, A.G., Mendelson, E.: Mathematical intuitionism. Introduction to proof theory (1990)
12. Otten, J.: leanCoP 2.0 and ileanCoP 1.2: high performance lean theorem proving in classical and intuitionistic logic (system descriptions). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 283–291. Springer, Heidelberg (2008). doi:10.1007/978-3-540-71070-7_23
13. Sutcliffe, G.: The TPTP problem library and associated infrastructure: the FOF and CNF parts, v3.5.0. J. Autom. Reasoning **43**(4), 337–362 (2009)

Semantics and Category Theory

A Light Modality for Recursion

Paula Severi

Department of Informatics, University of Leicester, Leicester, UK

Abstract. We investigate a modality for controlling the behaviour of recursive functional programs on infinite structures which is completely silent in the syntax. The latter means that programs do not contain "marks" showing the application of the introduction and elimination rules for the modality. This shifts the burden of controlling recursion from the programmer to the compiler.

To do this, we introduce a typed lambda calculus à la Curry with a silent modality and guarded recursive types. The typing discipline guarantees normalisation and can be transformed into an algorithm which infers the type of a program.

1 Introduction

The quest of finding a typing discipline that guarantees that functions on coinductive data types are productive has prompted a variety of works that rely on a modal operator [1,3,6,7,15,17,23]. All these works except for Nakano's [17] have explicit constructors and destructors in the syntax of programs [6,7,15,23]. This has the advantage that type inference is easy but it has the disadvantage that they do not really liberate the programmer from the task of controlling recursion since one has to know when to apply the introduction and elimination rules for the modal operator. As far as we know, decidability of type inference for Nakano's type system remains an open problem.

In this paper we consider the pure functional part of a type system studied previously [24]. This is a typed lambda calculus which has the advantage of having a modal operator silent in the syntax of programs without resorting to a subtyping relation as Nakano while keeping the nice properties of subject reduction and normalisation. We show that the type inference problem is decidable for the system under consideration. Even without a subtyping relation, this variant of the modal operator is still challenging to deal with because it is intrinsically non-structural, not corresponding to any expression form in the calculus.

Apart from the modal operator, we also include guarded recursive types which generalize the recursive equation $\mathtt{Str}_t = t \times \bullet\mathtt{Str}_t$ for streams [15,23]. This allows us to type productive functions on streams such as

$$\mathtt{skip}\ xs = \langle \mathtt{fst}\ xs, \mathtt{skip}\ (\mathtt{snd}\ (\mathtt{snd}\ xs))\rangle$$

which deletes the elements at even positions of a stream using the type $\mathtt{Str}_{\mathtt{Nat}} \to \mathtt{Str2}_{\mathtt{Nat}}$ where $\mathtt{Str2}_t = t \times \bullet\bullet\,\mathtt{Str2}_t$.

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 499–516, 2017.
DOI: 10.1007/978-3-662-54458-7_29

Lazy functional programming is acknowledged as a paradigm that fosters software modularity [12] and enables programmers to specify computations over possibly infinite data structures in elegant and concise ways. We give some examples that show how modularization and compositionality can be achieved using the modal operator. An important recursive pattern used in functional programming for modularisation is `foldr` defined by

$$\texttt{foldr}\ f\ xs = f\ (\texttt{fst}\ xs)\ (\texttt{foldr}\ f\ (\texttt{snd}\ xs))$$

The type of `foldr` which is $(t \to \bullet s \to s) \to \texttt{Str}_t \to s$, is telling us what is the safe way to build functions. While it is possible to type

$$\texttt{iterate}\ f = \texttt{foldr}\ (\lambda xy.\langle f\ x, y\rangle)$$

by assigning the type $t \to t$ to f, and assuming $s = \texttt{Str}_t$, it is not possible to type the unproductive function $\texttt{foldr}\ (\lambda xy.y)$. This is because $\lambda xy.y$ does not have type $(t \to \bullet s \to s)$.

In spite of the fact that $\texttt{Str}_{\texttt{Nat}} \to \texttt{Str2}_{\texttt{Nat}}$ is the type of `skip` with the minimal number of bullets, it does not give enough information to the programmer to know that the composition of `skip` with itself is still typeable. In order to assist the user in the task of modularization, we need to infer a more general type for `skip`, which would look like

$$\texttt{Stream}(N, X) \to \texttt{Stream}(N + 1, X)$$

where $\texttt{Stream}(N, X) = X \times \bullet^N \texttt{Stream}(N, X)$ and N and X are integer and type variable, respectively. It is clear now that from the above type, a programmer can deduce that it is possible to do the composition of `skip` with itself.

Contributions and Outline. Section 2 defines the typed lambda calculus $\lambda^{\bullet}_{\to}$ with a silent modal operator and proves subject reduction and normalisation. Section 3 gives an adequate denotational semantics for $\lambda^{\bullet}_{\to}$ in the topos of trees as a way of linking our system to the work by Birkedal et al. [2]. Section 4 shows the most important contribution of this paper which is decidability of the type inference problem for $\lambda^{\bullet}_{\to}$. This problem is solved by an algorithm which has the interesting feature of combining unification of types with integer linear programming. Sections 5 and 6 discuss related and future work, respectively.

2 A Light Modality for Typed Lambda Calculus

The syntax for *expressions* and *types* is given by the following grammars.

$e ::=^{ind}$		**Expression**
	x	(variable)
|	$\lambda x.e$	(abstraction)
|	ee	(application)
|	$\langle e, e\rangle$	(pair)
|	$(\texttt{fst}\ e)$	(first)
|	$(\texttt{snd}\ e)$	(second)

$t ::=^{coind}$		**Pseudo-type**
	X	(type variable)
|	$t \times t$	(product)
|	$t \to t$	(arrow)
|	$\bullet t$	(delay)

In addition to the usual constructs of the λ-calculus, expressions include pairs. We do not need a primitive constant for the fixed point operator because it can be expressed and typed inside the language. Expressions are subject to the usual conventions of the λ-calculus. In particular, we assume that the bodies of abstractions extend as much as possible to the right, that applications associate to the left, and we use parentheses to disambiguate the notation when necessary.

The syntax of *pseudo-types* is defined co-inductively. A type is a possibly infinite tree, where each internal node is labelled by a type constructor $\to$, $\times$ or $\bullet$ and has as many children as the arity of the constructor. The leaves of the tree (if any) are labelled by either type variables or $\mathtt{end}$. We use a co-inductive syntax to describe infinite data structures (such as streams). The syntax for pseudo-types include the types of the simply typed lambda calculus, *arrows* and *products* and the *delay* type constructor $\bullet$ [17]. An expression of type $\bullet t$ denotes a value of type t that is available "at the next moment in time". This constructor is key to control recursion and attain normalisation.

Definition 1 (Types). *We say that a pseudo-type t is*

1. regular *if its tree representation has finitely many distinct sub-trees.*
2. guarded *if every infinite path in its tree representation has infinitely many $\bullet$'s.*
3. *a* type *if it is regular and guarded.*

The regularity condition implies that we only consider types admitting a finite representation. It is equivalent to representing types with μ-notation and a strong equality which allows for an infinite number of unfoldings. This is also called the *equirecursive approach* since it views types as the unique solutions of recursive equations [10,19]. The existence and uniqueness of a solution satisfying condition 1 follow from known results (see [9] and also Theorem 7.5.34 of [5]). For example, there are unique types $\mathtt{Str}'_{\mathtt{Nat}}$, $\mathtt{Str}_{\mathtt{Nat}}$, and $\bullet^\infty$ that respectively satisfy the equations $\mathtt{Str}'_{\mathtt{Nat}} = \mathtt{Nat} \times \mathtt{Str}'_{\mathtt{Nat}}$, $\mathtt{Str}_{\mathtt{Nat}} = \mathtt{Nat} \times \bullet\mathtt{Str}_{\mathtt{Nat}}$, and $\bullet^\infty = \bullet\bullet^\infty$.

The guardedness condition intuitively means that not all parts of an infinite data structure can be available at once: those whose type is prefixed by a $\bullet$ are necessarily "delayed" in the sense that recursive calls on them must be deeper. For example, $\mathtt{Str}_{\mathtt{Nat}}$ is a type that denotes streams of natural numbers, where each subsequent element of the stream is delayed by one $\bullet$ compared to its predecessor. Instead $\mathtt{Str}'_{\mathtt{Nat}}$ is not a type: it would denote an infinite stream of natural numbers, whose elements are all available right away. If the types are written in μ-notation, the guardedness condition means that all occurrences of X in the body t of $\mu X.t$ are in the scope of a $\bullet$.

The type $\bullet^\infty$ is somehow degenerated in that it contains no actual data constructors. Unsurprisingly, we will see that non-normalising terms such as $\Omega = (\lambda x.x\ x)(\lambda x.x\ x)$ can only be typed with $\bullet^\infty$ (see Theorem 1). Without Condition 2, Ω could be given any type since the recursive pseudo-type $\mathtt{D} = \mathtt{D} \to t$ would become a type.

Sometimes we will write $\bullet^n t$ in place of $\underbrace{\bullet \cdots \bullet}_{n\text{-times}} t$.

We adopt the usual conventions regarding arrow types (which associate to the right) and assume the following precedence among type constructors: $\to$, $\times$, $\bullet$ with $\bullet$ having the highest precedence.

Expressions reduce according to a standard *call-by-name* semantics:

$$\frac{}{(\lambda x.e_1)\ e_2 \longrightarrow e_1[e_2/x]}\ [\text{R-BETA}] \qquad \frac{}{\texttt{fst}\ \langle e_1, e_2\rangle \longrightarrow e_1}\ [\text{R-FIRST}] \qquad \frac{}{\texttt{snd}\ \langle e_1, e_2\rangle \longrightarrow e_2}\ [\text{R-SECOND}] \qquad \frac{e \longrightarrow f}{\mathscr{E}[e] \longrightarrow \mathscr{E}[f]}\ [\text{R-CTXT}]$$

where the *evaluation contexts* are $\mathscr{E} ::= [\ ] \mid \mathscr{E}\, e \mid (\texttt{fst}\ \mathscr{E}) \mid (\texttt{snd}\ \mathscr{E})$. *Normal forms* are defined as usual as expressions that do not reduce.

The *type assignment system* $\lambda^{\bullet}_{\to}$ is defined by the following rules.

$$\frac{}{\Gamma, x : t \vdash x : t}\ [\text{AXIOM}] \qquad \frac{\Gamma \vdash e : t}{\Gamma \vdash e : \bullet t}\ [\bullet\text{I}] \qquad \frac{\Gamma, x : \bullet^n t \vdash e : \bullet^n s}{\Gamma \vdash \lambda x.e : \bullet^n (t \to s)}\ [\to\text{I}] \qquad \frac{\Gamma \vdash e_1 : \bullet^n (t \to s) \quad \Gamma \vdash e_2 : \bullet^n t}{\Gamma \vdash e_1 e_2 : \bullet^n s}\ [\to\text{E}]$$

$$\frac{\Gamma \vdash e_1 : \bullet^n t \qquad \Gamma \vdash e_2 : \bullet^n s}{\Gamma \vdash \langle e_1, e_2\rangle : \bullet^n (t \times s)}\ [\times\text{I}] \qquad \frac{}{\Gamma \vdash \texttt{fst} : t_1 \times t_2 \to t_1}\ [\times\text{E}_1] \qquad \frac{}{\Gamma \vdash \texttt{snd} : t_1 \times t_2 \to t_2}\ [\times\text{E}_2]$$

They are essentially the same as those for the simply typed lambda calculus except for two important differences. First, we now have an additional rule [$\bullet$I] for introducing the modality. This rule is unusual in the sense that the expression remains the same, i.e. we do not have a constructor for $\bullet$ at the level of expressions. Second, each rule allows for an arbitrary delay in front of the types of the entities involved. Intuitively, the number of $\bullet$'s represents the delay at which a value becomes available. So for example, rule [$\to$I] says that a function which accepts an argument x of type t delayed by n and produces a result of type s delayed by the same n has type $\bullet^n(t \to s)$, that is a function delayed by n that maps elements of t into elements of s. Crucially, it is not possible to *anticipate* a delayed value: if it is known that a value will only be available with delay n, then it will also be available with any delay $m \geq n$, but not earlier.

Using rule [$\bullet$I] and the recursive type $\texttt{D} = \bullet\texttt{D} \to t$, we can derive that the fixed point combinator $\texttt{fix} = \lambda y.(\lambda x.y\ (x\ x))(\lambda x.y\ (x\ x))$ has type $(\bullet t \to t) \to t$ by assigning the type $\texttt{D} \to t$ to the first occurrence of $\lambda x.y\ (x\ x)$ and $\bullet\texttt{D} \to t$ to the second one [17].

Let $\texttt{skip} = \texttt{fix}\ \lambda f x.\langle(\texttt{fst}\ x), f\ (\texttt{snd}\ (\texttt{snd}\ x))\rangle$ be the function that deletes the elements at even positions of a stream. In order to assign the type $\texttt{Str}_{\texttt{Nat}} \to \texttt{Str2}_{\texttt{Nat}}$ to skip, the variable f has to be delayed once and the first occurrence of snd has to be delayed twice. Note also that when typing the application $f\ (\texttt{snd}\ (\texttt{snd}\ x))$ the rule [$\to$E] is used with $n = 2$.

The following lemma expresses the fact that the type of an expression should be delayed as much as the types in the environment. The proof is by induction on the derivation.

Lemma 1 (Delay). *If* $\Gamma \vdash e : t$, *then* $\Gamma_1, \bullet\Gamma_2 \vdash e : \bullet t$ *for* $\Gamma_1, \Gamma_2 = \Gamma$.

For example, from $x : t \vdash \lambda y.x : s \to t$ we can deduce that $x : \bullet t \vdash \lambda y.x : \bullet(s \to t)$, but we cannot deduce $x : \bullet t \vdash \lambda y.x : s \to t$. The Delay Lemma is crucial for proving Inversion Lemma:

Lemma 2 (Inversion).

1. *If* $\Gamma \vdash x : t$, *then* $t = \bullet^n t'$ *and* $x \in \mathsf{dom}(\Gamma)$.
2. *If* $\Gamma \vdash \lambda x.e : t$, *then* $t = \bullet^n(t_1 \to t_2)$ *and* $\Gamma, x : \bullet^n t_1 \vdash e : \bullet^n t_2$.
3. *If* $\Gamma \vdash e_1 e_2 : t$, *then* $t = \bullet^n t_2$ *and* $\Gamma \vdash e_2 : \bullet^n t_1$ *and* $\Gamma \vdash e_1 : \bullet^n(t_1 \to t_2)$.
4. *If* $\Gamma \vdash \mathtt{fst} : t$, *then* $t = \bullet^n(t_1 \times t_2 \to t_1)$.
5. *If* $\Gamma \vdash \mathtt{snd} : t$, *then* $t = \bullet^n(t_1 \times t_2 \to t_2)$.

Proof. By case analysis and induction on the derivation. We only show Item 2 which is interesting because we need to shift the environment in time and apply Lemma 1. A derivation of $\Gamma \vdash \lambda x.e : t$ ends with an application of either $[\to\mathrm{I}]$ or $[\bullet\mathrm{I}]$. For the former case, the proof is immediate. If the last applied rule is $[\bullet\mathrm{I}]$, then $t = \bullet t'$ and $\Gamma \vdash \lambda x.e : t'$. By induction $t' = \bullet^n(t_1 \to t_2)$ and $\Gamma, x : \bullet^n t_1 \vdash e : \bullet^n t_2$. Hence $t = \bullet t' = \bullet^{n+1}(t_1 \to t_2)$ and
by Lemma 1 $\Gamma, x : \bullet^{n+1} t_1 \vdash e : \bullet^{n+1} t_2$.

An important consequence of Inversion Lemma is Subject Reduction:

Lemma 3 (Subject Reduction). *If* $\Gamma \vdash e : t$ *and* $e \longrightarrow e'$ *then* $\Gamma \vdash e' : t$.

Proof. By induction on the definition of $\longrightarrow$. We only do the case $(\lambda x.e_1)\ e_2 \longrightarrow e_1[e_2/x]$. Suppose $\Gamma \vdash (\lambda x.e_1)\ e_2 : t$. By Item 3 of Lemma 2

$$t = \bullet^n t_2 \qquad \Gamma \vdash e_2 : \bullet^n t_1 \qquad \Gamma \vdash (\lambda x.e_1) : \bullet^n(t_1 \to t_2)$$

It follows from Item 2 of Lemma 2 that $\Gamma, x : \bullet^n t_1 \vdash e_1 : \bullet^n t_2$. By applying Substitution Lemma (which follows by an easy induction on expressions), we deduce that $\Gamma \vdash e_1[e_2/x] : \bullet^n t_2$.

Neither Nakano's type system nor ours is closed under η-reduction. For example, $y : \bullet(t \to s) \vdash \lambda x.(y\ x) : (t \to \bullet s)$ but $y : \bullet(t \to s) \not\vdash y : (t \to \bullet s)$. The lack of subject reduction for η-reduction does not seem important in the context of lazy evaluation where programs are closed terms and only weak head normalised.

Theorem 1 (Normalisation) *[24,25].* *If* $\Gamma \vdash e : t$ *and* $t \neq \bullet^\infty$, *then* e *reduces (in zero or more steps) to a normal form.*

The proof of the above theorem follows from the fact that $\lambda^\bullet_\to$ is included in the type system of [24] and the latter is normalising [25]. Notice that there are normalising expressions that cannot be typed, for example $\lambda x.\Omega\mathbf{I}$, where $\Omega = (\lambda y.y\ y)(\lambda y.y\ y)$ and $\mathbf{I} = \lambda z.z$. In fact Ω has type $\bullet^\infty$ and by previous theorem it cannot have other types, and this implies that the application $\Omega\mathbf{I}$ has no type.

3 Denotational Semantics

This section gives a denotational semantics for $\lambda^{\bullet}_{\rightarrow}$ where types and expressions are interpreted as objects and morphisms in the *topos of trees* $\mathsf{Set}^{\omega^{op}}$ [2]. We give a self-contained description of this topos as a cartesian closed category for a reader familiar with λ-calculus. The *topos of trees* $\mathscr{S}$ has as objects A families of sets $A_1, A_2, \ldots$ indexed by positive integers, equipped with family of restrictions $r_i^A : A_{i+1} \to A_i$. Types will be interpreted as family of sets (not just sets). Intuitively the family represents better and better sets of approximants for the elements of that type. Arrows $f : A \to B$ are families of functions $f_i : A_i \to B_i$ obeying the naturality condition $f_i \circ r_i^A = r_i^B \circ f_{i+1}$.

$$\begin{array}{ccccccc} A_1 & \xleftarrow{r_1^A} & A_2 & \xleftarrow{r_2^A} & A_3 & & \cdots \\ {\scriptstyle f_1}\downarrow & & {\scriptstyle f_2}\downarrow & & {\scriptstyle f_3}\downarrow & & \\ B_1 & \xleftarrow[r_1^B]{} & B_2 & \xleftarrow[r_2^B]{} & B_3 & & \cdots \end{array}$$

We define $r_{ii}^A = id_A$ and $r_{ij}^A = r_j \circ \ldots \circ r_{i-1}$ for $1 \leq j < i$. Products are defined pointwise. Exponentials B^A have as components the sets:

$$(B^A)_i = \{(f_1, \ldots, f_i) \mid f_j : A_j \to B_j \text{ and } f_j \circ r_j^A = r_j^B \circ f_{j+1}\}$$

and as restrictions $r_i^{A \Rightarrow B}(f_1, \ldots f_{i+1}) = (f_1, \ldots, f_i)$. We define $\mathsf{eval} : B^A \times A \to B$ as $\mathsf{eval}_i((f_1, \ldots, f_i), a) = f_i(a)$ and $\mathsf{curry}(f) : C \to B^A$ for $f : C \times A \to B$ as

$$\mathsf{curry}(f)_i(c) = (g_1, \ldots, g_i)$$

where $g_j(a) = f_j(r_{ij}^A(c), a)$ for all $a \in A_j$ and $1 \leq j \leq i$. The functor $\blacktriangleright : \mathscr{S} \to \mathscr{S}$ is defined on objects as $(\blacktriangleright\ A)_1 = \{*\}$ and $(\blacktriangleright\ A)_{i+1} = A_i$ where $r_1^{\blacktriangleright A} =!$ and $r_{i+1}^{\blacktriangleright A} = r_i^A$ and on arrows $(\blacktriangleright\ f)_1 = id_{\{*\}}$ and $(\blacktriangleright\ f)_{i+1} = f_i$. We write $\blacktriangleright^n$ for the n-times composition of $\blacktriangleright$.

The natural transformation $\mathsf{next}_A : A \to \blacktriangleright\ A$ is given by $(\mathsf{next}_A)_1 =!$ and $(\mathsf{next}_A)_{i+1} = r_i^A$ which can easily be extended to a natural transformation $\mathsf{next}_A^n : A \to \blacktriangleright^n\ A$. It is not difficult to see that there are isomorphisms $\theta : \blacktriangleright\ A \times \blacktriangleright\ B \to \blacktriangleright\ (A \times B)$ and $\xi : (\blacktriangleright\ B)^{(\blacktriangleright A)} \to \blacktriangleright\ (B^A)$ which are also natural. These can also be easily extended to isomorphisms $\theta^n : \blacktriangleright^n\ A \times \blacktriangleright^n\ B \to \blacktriangleright^n\ (A \times B)$ and $\xi^n : (\blacktriangleright^n\ B)^{(\blacktriangleright^n A)} \to \blacktriangleright^n\ (B^A)$.

A type t is interpreted as a functor $[\![t]\!] \in (\mathscr{S}^{op} \times \mathscr{S})^k \to \mathscr{S}$ by fixing a superset $X_1 \ldots X_k$ of its free variables. The mixed variance is a way of solving the problem of the contra-variance and the functoriality of $[\![t]\!]$ since variables can appear positively and negatively [7].

$$\begin{array}{ll} [\![X]\!]\overrightarrow{(V,W)} & = W_j \text{ if } X = X_j \text{ for } 1 \leq j \leq k \\ [\![t \times s]\!]\overrightarrow{(V,W)} & = [\![t]\!]\overrightarrow{(V,W)} \times [\![s]\!]\overrightarrow{(V,W)} \\ [\![t \to s]\!]\overrightarrow{(V,W)} & = [\![s]\!]\overrightarrow{(V,W)}^{[\![t]\!]\overrightarrow{(W,V)}} \\ [\![\bullet t]\!] & = \blacktriangleright \circ [\![t]\!] \end{array}$$

In order to justify that the interpretation is well-defined, it is necessary to view the above definition as indexed sets and we do induction on the pair $(i, rank(t))$ taking the lexicographic order where $rank(t)$ is defined by $rank(\bullet t) = 0$ and $rank(t \times s) = rank(t \to s) = max(rank(t), rank(s)) + 1$. By writing the indices explicitly for $[\![\bullet t]\!]$, we obtain

$$([\![\bullet t]\!]\overrightarrow{(V,W)})_1 = \{*\} \qquad ([\![\bullet t]\!]\overrightarrow{(V,W)})_{i+1} = ([\![t]\!]\overrightarrow{(V,W)})_i$$

where the index decreases. For $t \times s$ and $t \to s$, the interpretation at i is defined in terms of the interpretations of t and s at i, so the rank decreases.

Alternatively, if we represent types in μ-notation, the interpretation can be defined by induction on the type by adding the case:

$$[\![\mu X.t]\!]\overrightarrow{(V,W)} = \mathsf{Fix}(F) \text{ where } F(V_1, W_1) = [\![t]\!](\overrightarrow{(V,W)}, (V_1, W_1))$$
$$\text{and } \mathsf{Fix}(F) \text{ is the unique } A \text{ such that } F(A, A) \cong A$$

The existence of the fixed point $\mathsf{Fix}(F)$ follows from [2, Section 4.5] since F is *locally contractive*.

As it is common in categorical semantics, the interpretation is not defined on lambda terms in isolation but on typing judgements. In order to define terms as morphisms, we need the context and the type to specify their domain and co-domain. Typing contexts $\Gamma = x_1 : t_1, \ldots x_k : t_k$ are interpreted as $[\![t_1]\!] \times \ldots \times [\![t_k]\!]$. The interpretation of typed expressions $[\![\Gamma \vdash e : t]\!] : [\![\Gamma]\!] \to [\![t]\!]$ is defined by induction on e (using Inversion Lemma):

$$\begin{array}{ll}
[\![\Gamma \vdash x : \bullet^n t]\!] & = \mathsf{next}^n \circ \pi_j \text{ if } x = x_j \text{ and } t_j = t \text{ and } 1 \leq j \leq k \\
[\![\Gamma \vdash e_1 e_2 : \bullet^n s]\!] & = \mathsf{eval} \circ \langle (\xi^n)^{-1} \circ [\![\Gamma \vdash e_1 : \bullet^n (t \to s)]\!], [\![\Gamma \vdash e_2 : \bullet^n t]\!] \rangle \\
[\![\Gamma \vdash \lambda x.e : \bullet^n (t \to s)]\!] & = \xi^n \circ \mathsf{curry}([\![\Gamma, x : \bullet^n t \vdash e : \bullet^n s]\!]) \\
[\![\Gamma \vdash \langle e_1 , e_2 \rangle : \bullet^n (t \times s)]\!] & = (\theta^n) \circ \langle [\![\Gamma \vdash e_1 : \bullet^n t]\!], [\![\Gamma \vdash e_2 : \bullet^n s]\!] \rangle \\
[\![\Gamma \vdash (\mathtt{fst}\ e) : \bullet^n t]\!] & = \pi_1 \circ (\theta^n)^{-1} \circ [\![\Gamma \vdash e : \bullet^n (t \times s)]\!] \\
[\![\Gamma \vdash (\mathtt{snd}\ e) : \bullet^n t]\!] & = \pi_2 \circ (\theta^n)^{-1} \circ [\![\Gamma \vdash e : \bullet^n (t \times s)]\!]
\end{array}$$

Lemma 4 (Semantic Substitution).

$$[\![\Gamma, x : \bullet^n t \vdash e_1 : \bullet^n s]\!] \circ \langle id, [\![\Gamma \vdash e_2 : \bullet^n t]\!] \rangle = [\![\Gamma \vdash e_1[e_2/x] : \bullet^n s]\!]$$

Proof. This follows by induction on e_1. We only show the case $e_1 = x$. By Inversion Lemma $s = \bullet^m t$. Then,

$$[\![\Gamma, x : \bullet^n t \vdash e_1 : \bullet^{m+n} t]\!] \circ \langle id, [\![\Gamma \vdash e_2 : \bullet^n t]\!] \rangle = \mathsf{next}^m \circ [\![\Gamma \vdash e_2 : \bullet^n t]\!] = [\![\Gamma \vdash e_2 : \bullet^{n+m} t]\!]$$

The last equality follows from a semantic delay lemma.

Theorem 2 (Soundness). *If* $\Gamma \vdash e : t$ *and* $e \longrightarrow e'$ *then* $[\![\Gamma \vdash e : t]\!] = [\![\Gamma \vdash e' : t]\!]$.

Proof. We show the case of [R-BETA]. Let $v_1 = [\![\Gamma, x : \bullet^n t \vdash e_1 : \bullet^n s]\!]$ and $v_2 = [\![\Gamma \vdash e_2 : \bullet^n t]\!]$.

$$[\![\Gamma \vdash (\lambda x.e_1)e_2 : \bullet^n s]\!] = \mathsf{eval} \circ \langle (\xi^n)^{-1} \circ \xi^n \circ \mathsf{curry}(v_1), v_2 \rangle = v_1 \circ \langle id, v_2 \rangle$$
$$= [\![\Gamma \vdash e_1[e_2/x] : \bullet^n s]\!] \text{ by Lemma 4}$$

The denotational semantics of $\lambda^{\bullet}_{\rightarrow}$ in the topos of trees can be generalised to $\mathsf{Set}^{A^{op}}$ for a set A equipped with a well-founded relation.

4 A Type Inference Algorithm

In this section, we define a type inference algorithm for $\lambda^{\bullet}_{\rightarrow}$. Apart from the usual complications that come from having no type declarations, the difficulty of finding an appropriate type inference algorithm for $\lambda^{\bullet}_{\rightarrow}$ is due to the fact that the expressions do not have a constructor and destructor for $\bullet$. We do not know which sub-expressions need to be delayed as illustrated by the type derivation of `fix` where the first occurrence of $(\lambda x.y\ (x\ x))$ has a different derivation from the second one since $[\bullet\mathrm{I}]$ is applied in different places [17]. Even worse, in case a sub-expression has to be delayed, we do not know how many times needs to be delayed to be able to type the whole expression.

The type inference algorithm infers *meta-types* which are a generalization of types where the $\bullet$ can be exponentiated with integer expressions, e.g. $\bullet^{N_1} X \rightarrow \bullet^{N_2 - N_1} X$. The algorithm proceeds in several stages. The first stage generates a meta-type T_0 and a set $\mathscr{C}_0$ of meta-type constraints from a given closed expression e. Secondly, the unification algorithm transforms $\mathscr{C}_0$ into a set $\mathscr{C}$ of recursive equations and it simultaneously generates a set $\mathscr{E}$ of integer constraints to guarantee that the meta-types have non-negative exponents. Thirdly, a set $\mathsf{gC}(\mathscr{C})$ of integers constraints is computed to ensure that the solution is guarded. If $\mathscr{E} \cup \mathsf{gC}(\mathscr{C})$ is solvable then e is typable and its type is obtained by substituting T_0 by the solutions of $\mathscr{C}$ and $\mathscr{E} \cup \mathsf{gC}(\mathscr{C})$.

4.1 Meta-Types

The syntax for *pseudo meta-types* is defined below. A pseudo meta-type can contain type variables and (non-negative) integer expressions with variables. In this syntax, $\bullet t$ is written as $\bullet^1 t$. We identify $\bullet^0 t$ with t and $\bullet^E \bullet^{E'} t$ with $\bullet^{E+E'} t$. We define a *meta-type* as a pseudo meta-type that is regular and guarded.

$E ::=^{ind}$		**Integer Expression**	$T ::=^{coind}$		**Pseudo Meta-type**
	N	(integer variable)		X	(type variable)
|	$\mathtt{n}$	(integer number)	|	$T \times T$	(product)
|	$E + E$	(addition)	|	$T \rightarrow T$	(arrow)
|	$E - E$	(substraction)	|	$\bullet^E T$	(delay)

Let τ be a finite mapping from type variables to meta-types, denoted as $\{X_1 \mapsto T_1, \ldots, X_n \mapsto T_n\}$, and let ρ be a finite mapping from integer variables

to integer expressions, denoted as $\{N_1 \mapsto E_1, \ldots, N_m \mapsto E_m\}$. Let also $\sigma = \tau \cup \rho$. We define the substitution on a meta-types and an integer expression as follows.

$$\begin{array}{lll} N\rho & = E \text{ if } N \mapsto E \in \rho & X\sigma & = T \text{ if } X \mapsto T \in \tau \\ \mathtt{n}\rho & = \mathtt{n} & (T_1 \to T_2)\sigma & = T_1\sigma \to T_2\sigma \\ (E_1 + E_2)\rho & = E_1\rho + E_2\rho & (T_1 \times T_2)\sigma & = T_1\sigma \times T_2\sigma \\ (E_1 - E_2)\rho & = E_1\rho - E_2\rho & (\bullet^E T)\sigma & = \bullet^{E\rho} T\sigma \end{array}$$

We say that $\sigma = \tau \cup \rho$ is a *ground substitution* if ρ maps all integer variables into natural numbers. We say that T is *ground* if it contains no integer variables and all the exponents of $\bullet$ are natural numbers. We identify a ground type with the type obtained from replacing the type constructor $\bullet^n$ in the syntax of meta-types with n consecutive $\bullet$'s in the syntax of types.

Definition 2 (Constraints). *A* meta-type constraint *is an equation* $T \stackrel{?}{=} T'$ *between finite meta-types. An integer constraint is either* $E \stackrel{?}{=} E'$ *or* $E \stackrel{?}{\geq} E'$ *or* $E \stackrel{?}{<} E'$.

Moreover, $\sigma \models T \stackrel{?}{=} T'$ means that $T\sigma = T'\sigma$. Similarly, we define $\rho \models E \stackrel{?}{=} E'$, $\rho \models E \stackrel{?}{\geq} E'$ and $\rho \models E \stackrel{?}{<} E'$. This notation extends to a set $\mathscr{C}$ (or $\mathscr{E}$) of constraints in the obvious way.

We say that $\mathscr{C}$ is *substitutional* if $\mathscr{C} = \{X_1 \stackrel{?}{=} T_1, \ldots, X_n \stackrel{?}{=} T_n\}$ and all variables $X_1, \ldots, X_n$ are pairwise different. Since a substitutional $\mathscr{C}$ is a set of recursive equations where $T_1, \ldots, T_n$ are finite meta-types, there exists a unique solution $\tau_{\mathscr{C}}$ such that $\tau_{\mathscr{C}}(X_i)$ is regular for all $1 \leq i \leq n$ [9], [5, Theorem 7.5.34]. Note that the unicity of the solution of a set of recursive equations would not be guaranteed if we were following the iso-recursive approach which allows only for a finite number of unfoldings $\mu X.t = t\{X \mapsto \mu X.t\}$ [4,5].

We say that $\mathscr{E}$ *grounds* T if $T\rho$ is ground for all ground substitutions ρ such that $\rho \models \mathscr{E}$. For example, $N_1 \geq N_2$ grounds $(\bullet^{N_1 - N_2} X)$ but $N_1 \leq N_2$ does not.

4.2 Constraint Typing Rules

Table 1 defines the *constraint typing rules*. We assume that Δ only contains declarations of the form $x : X$. This is needed for the proof of Item 2 of Theorem 3. If instead of generating the fresh variable x in $[\to\mathrm{I}]$ we directly put $\bullet^N X_1$ in the context, then we would not know what value to assign N in the case of $e = x$ unless we look at the rest of the type derivation. For example, consider $x : \bullet^5 \mathtt{Nat} \vdash x : \bullet^7 \mathtt{Nat}$. Then N should be assigned the value 3 and not 5 if later we derive that $\lambda x.x : \bullet^2(\bullet^3 X \to \bullet^5 X)$.

Note that given an expression e, it is always possible to derive Δ and $\mathscr{C}$ such that $\Delta \vdash e : T \mid \mathscr{C}$ and the set $\mathscr{C}$ contains only constraints between finite meta-types.

Theorem 3 (Correctness of Constraint Typing). *Let* $\Delta \vdash e : T \mid \mathscr{C}$ *and* $\Gamma = x_1 : t_1, \ldots, x_m : t_m$ *and* $\Delta = x_1 : X_1, \ldots, x_m : X_m$.

Table 1. Constraint typing rules

$$[\text{AXIOM}]\ \frac{}{\Delta, x : X \vdash x : \bullet^N X \mid \emptyset} \qquad [\to\text{I}]\ \frac{\Delta, x : X \vdash e : T \mid \mathscr{C} \qquad X, X_1, X_2, N \text{ are fresh}}{\Delta \vdash \lambda x.e : \bullet^N(X_1 \to X_2) \mid \mathscr{C} \cup \{X \stackrel{?}{=} \bullet^N X_1, T \stackrel{?}{=} \bullet^N X_2\}}$$

$$[\to\text{E}]\ \frac{\Delta \vdash e_1 : T_1 \mid \mathscr{C}_1 \qquad \Delta \vdash e_2 : T_2 \mid \mathscr{C}_2 \qquad X_1, X_2, N \text{ are fresh}}{\Delta \vdash e_1 e_2 : \bullet^N X_2 \mid \mathscr{C}_1 \cup \mathscr{C}_2 \cup \{\bullet^N(X_1 \to X_2) \stackrel{?}{=} T_1, \bullet^N X_1 \stackrel{?}{=} T_2\}}$$

$$[\times\text{I}]\ \frac{\Delta \vdash e_1 : T_1 \mid \mathscr{C}_1 \qquad \Delta \vdash e_2 : T_2 \mid \mathscr{C}_2 \qquad X_1, X_2, N \text{ are fresh}}{\Delta \vdash \langle e_1, e_2 \rangle : \bullet^N(X_1 \times X_2) \mid \mathscr{C}_1 \cup \mathscr{C}_2 \cup \{\bullet^N X_1 \stackrel{?}{=} T_1, \bullet^N X_2 \stackrel{?}{=} T_2\}}$$

$$[\times\text{E}_1]\ \frac{N, X_1, X_2 \text{ are fresh}}{\Delta \vdash \texttt{fst} : \bullet^N(X_1 \times X_2 \to X_1) \mid \emptyset} \qquad [\times\text{E}_2]\ \frac{N, X_1, X_2 \text{ are fresh}}{\Delta \vdash \texttt{snd} : \bullet^N(X_1 \times X_2 \to X_2) \mid \emptyset}$$

1. *Let $\Delta\sigma$ and $T\sigma$ be ground. If $\sigma \models \mathscr{C}$ then $\Delta\sigma \vdash e : T\sigma$.*
2. *If $\Gamma \vdash e : t$ then there exists a ground substitution $\sigma \supseteq \{X_1 \mapsto t_1, \ldots, X_m \mapsto t_m\}$ such that $\sigma \models \mathscr{C}$ and $T\sigma = t$ and $\mathsf{dom}(\sigma) \setminus \{X_1, \ldots, X_m\}$ is the set of fresh variables in the derivation of $\Gamma \vdash e : t$.*

Proof. Items 1 and 2 follow by induction on e. For Item 1, we prove the case of the abstraction. It follows from induction hypothesis that $\Delta\sigma, x : T_1\sigma \vdash e : T_2\sigma$. Since $\sigma \models \{T_1 \stackrel{?}{=} \bullet^N X_1, T_2 \stackrel{?}{=} \bullet^N X_2\}$, it is obvious that we also have that $\Delta\sigma, x : \bullet^{N\sigma} X_1\sigma \vdash e : \bullet^{N\sigma} X_2\sigma$. We can, then, apply $[\to\text{I}]$ to conclude that the abstraction has type $\bullet^{N\sigma}(X_1\sigma \to X_2\sigma)$ from the context $\Delta\sigma$.

For Item 2 we prove a few cases. Suppose $e = x$. By Inversion Lemma, we have that $x = x_i$ and $t = \bullet^n t_i$ for some i. We define $\sigma = \{X_1 \mapsto t_1, \ldots, X_m \mapsto t_m\} \cup \{N \mapsto n\}$. It is easy to see that $\Delta\sigma = \Gamma$ and $t = (\bullet^N X_i)\sigma$.

Suppose $e = e_1 e_2$. By Inversion Lemma, $t = \bullet^n t_2$ and $\Gamma \vdash e_2 : \bullet^n t_1$ and $\Gamma \vdash e_1 : \bullet^n(t_1 \to t_2)$. By induction hypotheses there are $\sigma_1, \sigma_2 \supseteq \{X_1 \mapsto t_1, \ldots, X_m \mapsto t_m\}$ such that $\sigma_1 \models \mathscr{C}_1$ and $\sigma_2 \models \mathscr{C}_2$ and $T_1\sigma_1 = \bullet^n(t_1 \to t_2)$ and $T_1\sigma_2 = \bullet^n t_2$. Since, by induction hypotheses, $\mathsf{dom}(\sigma_1) \setminus \{X_1, \ldots, X_m\}$ is the set of fresh variables in the derivation of $\Gamma \vdash e_1 : t$ and $\mathsf{dom}(\sigma_2) \setminus \{X_1, \ldots, X_m\}$ is the set of fresh variables in the derivation of $\Gamma \vdash e_2 : t$, we have that $\sigma_1 \cup \sigma_2$ is a function. We can now define the substitution σ as $(\sigma_1 \cup \sigma_2)\{N \mapsto n\}\{X_1 \mapsto t_1\}\{X_2 \mapsto t_2\}$.

4.3 Unification Algorithm

The *unification algorithm* is defined in Table 2. Given a set $\mathscr{C}$ of constraints, it returns a set of pairs $(\mathscr{C}', \mathscr{E})$ such that $\mathscr{C}'$ is substitutional and $\mathscr{E}$ grounds the meta-types in $\mathscr{C}'$. The extra argument $\mathscr{D}$ keeps tracks of the "visited type

constraints" and guarantees termination. The function unify does not return only one solution but a set of solutions. If $\bullet^N X \stackrel{?}{=} \bullet^{N'} X'$ then there are two ways of solving this equation: either $N \geq N'$ and $X' = \bullet^{N-N'} X$ or $N' > N$ and $X = \bullet^{N'-N} X'$. We use $\oplus$ for the disjoint union to guarantee that the type constraint that is being processed is removed from the set $\mathscr{C}$. The case $\mathscr{C}' \oplus \{\bullet^E X \stackrel{?}{=} \bullet^{E'} T\}$ assumes T is of the form T_1 op T_2. It is clear that if $E' < E$, then the unification does not have a solution, e.g. $\bullet^{-2} X \stackrel{?}{=} T_1$ op T_2. The case for $\mathscr{C}' \oplus \{T_1 \stackrel{?}{=} T_2, X \stackrel{?}{=} T_2\}$ is crucial for getting a substitutional set. By adding $T_1 \stackrel{?}{=} T_2$ and removing $X \stackrel{?}{=} T_2$ from the set $\mathscr{C}$, we obtain an equivalent set of constraints that "reduces" the set of constraints for X. Since $\mathscr{D}$ already contains $T_1 \stackrel{?}{=} T_2$, this constraint will not be added again avoiding non-termination. The unification algorithm for the simply typed lambda calculus solves the problem of termination in this case by reducing the number of variables, i.e. checks if $X \notin T_1$ and then, substitutes the variable x by T_1 in the remaining set of constraints. With recursive types, however, we do not perform the occur check and the number of variables may not decrease since the variable X may not disappear after substituting X by T (because X occurs in T). In order to decrease the number of variables, we could perhaps substitute X by $FIX\ (\lambda X.T_1)$ where FIX gives the solution of the recursive equation $X = T_1$ as a possible *infinite* tree. But the problem of guaranteeing termination would still be present if the meta-type constraints are allowed to be infinite since the size of the constraints may not decrease in some cases. We would also have a similar problem with termination if we use multi-equations and a rewrite relation instead of giving a function such as unify [20].

We start the algorithm by invoking $\mathsf{unify}(\mathscr{C}_0, \mathscr{C}_0)$ where the first and second argument are the same. In the remaining recursive calls, the second argument either remains the same or it is extended with the type constraint that has been just processed.

The size $|\ \mathscr{C}\ |$ of a set of constraints is the sum of the number of type variables and type constructors in the left hand side of the type constraints. Since the type constraints are finite, the size is always finite. We define

$$\mathsf{SubT}(\mathscr{C}_0) = \{T \mid S_1 \stackrel{?}{=} S_2 \in \mathscr{C}_0 \text{ and either } S_1 \text{ or } S_2 \text{ contains } T\}$$
$$\mathsf{SubC}(\mathscr{C}_0) = \{T_1 \stackrel{?}{=} T_2 \mid T_1, T_2 \in \mathsf{SubT}(\mathscr{C}_0)\}$$

In the evaluation of $\mathsf{unify}(\mathscr{C}_0, \mathscr{C}_0)$, the argument $\mathscr{D}$ of the recursive calls satisfies $\mathscr{C}_0 \subseteq \mathscr{D} \subseteq \mathsf{SubC}(\mathscr{C}_0)$.

Theorem 4 (Termination and Correctness of Unification). *Let* $\mathscr{D} \subseteq \mathsf{SubC}(\mathscr{C}_0)$.

1. $\mathsf{unify}(\mathscr{C}, \mathscr{D})$ *terminates*
2. *If* $(\mathscr{C}_0, \mathscr{E}_0) \in \mathsf{unify}(\mathscr{C}, \mathscr{D})$ *then* $\mathscr{C}_0$ *is substitutive.*
3. *If* $\sigma \models \mathscr{C}$ *then there exists* $(\mathscr{C}_0, \mathscr{E}_0) \in \mathsf{unify}(\mathscr{C}, \mathscr{D})$ *such that* $\sigma \models \mathscr{C}_0$ *and* $\sigma \models \mathscr{E}_0$.

Table 2. Unification algorithm

$$
\begin{aligned}
&\mathsf{unify}(\mathscr{C},\mathscr{D}) = \text{if } \mathscr{C} \text{ is substitutional then } \{(\mathscr{C},\emptyset)\}\\
&\qquad\text{else Case } \mathscr{C} \text{ of}\\
&\qquad\mathscr{C}' \oplus \{T \stackrel{?}{=} T\} \Rightarrow \mathsf{unify}(\mathscr{C}',\mathscr{D})\\
&\qquad\mathscr{C}' \oplus \{T \stackrel{?}{=} X\} \Rightarrow \mathsf{unify}(\mathscr{C}' \cup \{X \stackrel{?}{=} T\}, \mathscr{D} \cup \{X \stackrel{?}{=} T\})\\
&\qquad\mathscr{C}' \oplus \{X \stackrel{?}{=} T_1, X \stackrel{?}{=} T_2\} \Rightarrow\\
&\qquad\quad\text{if } T_1 \stackrel{?}{=} T_2 \in \mathscr{D} \text{ then } \mathsf{unify}(\mathscr{C}' \cup \{X \stackrel{?}{=} T_1\}, \mathscr{D})\\
&\qquad\quad\text{else } \mathsf{unify}(\mathscr{C}' \cup \{X \stackrel{?}{=} T_1, T_1 \stackrel{?}{=} T_2\}, \mathscr{D} \cup \{T_1 \stackrel{?}{=} T_2\})\\
&\qquad\mathscr{C}' \oplus \{\bullet^E X \stackrel{?}{=} \bullet^{E'} X'\} \Rightarrow\\
&\qquad\quad\{(\mathscr{C}_0, \mathscr{E}_0 \cup \{E' \geq E\}) \mid (\mathscr{C}_0, \mathscr{E}_0) \in \mathsf{unify}(\mathscr{C}_1, \mathscr{D})\} \cup\\
&\qquad\quad\{(\mathscr{C}_0, \mathscr{E}_0 \cup \{E > E'\}) \mid (\mathscr{C}_0, \mathscr{E}_0) \in \mathsf{unify}(\mathscr{C}_2, \mathscr{D})\}\\
&\qquad\quad\text{where } \mathscr{C}_1 = \mathscr{C}' \cup \{X \stackrel{?}{=} \bullet^{E'-E} X'\} \text{ and } \mathscr{C}_2 = \mathscr{C}' \cup \{X' \stackrel{?}{=} \bullet^{E-E'} X\}\\
&\qquad\mathscr{C} \oplus \{\bullet^E X \stackrel{?}{=} \bullet^{E'} T\} \Rightarrow\\
&\qquad\quad\{(\mathscr{C}_0, \mathscr{E}_0 \cup \{E' \geq E\}) \mid (\mathscr{C}_0, \mathscr{E}_0) \in \mathsf{unify}(\mathscr{C}_1, \mathscr{D})\}\\
&\qquad\quad\text{where } \mathscr{C}_1 = \mathscr{C} \cup \{X \stackrel{?}{=} \bullet^{E'-E} T\}\\
&\qquad\mathscr{C} \oplus \{\bullet^{E'} T \stackrel{?}{=} \bullet^E X\} \Rightarrow \mathsf{unify}(\mathscr{C} \cup \{\bullet^E X \stackrel{?}{=} \bullet^{E'} T\}, \mathscr{D})\\
&\qquad\mathscr{C} \oplus \{\bullet^E (T_1 \,\mathsf{op}_1\, T_2) \stackrel{?}{=} \bullet^{E'} (T_1' \,\mathsf{op}_2\, T_2')\} \Rightarrow\\
&\qquad\quad\text{if } \mathsf{op}_1 \neq \mathsf{op}_2 \text{ then fail}\\
&\qquad\quad\text{else } \{(\mathscr{C}_0, \mathscr{E}_0 \cup \{E \stackrel{?}{=} E'\}) \mid (\mathscr{C}_0, \mathscr{E}_0) \in \mathsf{unify}(\mathscr{C}', \mathscr{D})\}\\
&\qquad\quad\text{where } \mathscr{C}' = \mathscr{C} \cup \{T_1 \stackrel{?}{=} T_1', T_2 \stackrel{?}{=} T_2'\}
\end{aligned}
$$

4. *If* $(\mathscr{C}_0, \mathscr{E}_0) \in \mathsf{unify}(\mathscr{C}, \mathscr{D})$ *and* ρ *is a ground substitution such that* $\rho \models \mathscr{E}_0$ *then* $\rho \cup \tau_{\mathscr{C}_0} \models \mathscr{C}$ *and* $\rho(\tau_{\mathscr{C}_0}(X))$ *is ground for all* X.

Proof. In order to prove Item 1, observe that the second argument increases or remains the same in each recursive call. In the cases it remains the same, it is easy to see that $|\,\mathscr{C}\,|$ decreases. Since $\mathscr{D} \subseteq \mathsf{SubC}(\mathscr{C}_0)$, we have that

$$(|\,\mathsf{SubC}(\mathscr{C}_0)\,| - |\,\mathscr{D}\,|, |\,\mathscr{C}\,|)$$

decreases with each recursive call and hence, the unification algorithm terminates.

Items 2, 3 and 4 follow by induction on the number of recursive calls. For Item 3 we prove the case $\mathscr{C} = \mathscr{C}' \oplus \{\bullet^E (T_1 \,\mathsf{op}\, T_2) \stackrel{?}{=} \bullet^{E'} (T_1' \,\mathsf{op}\, T_2')\}$. It follows from $\sigma \models \mathscr{C}$ that $\sigma \models \mathscr{C}' \cup \{T_1 \stackrel{?}{=} T_1', T_2 \stackrel{?}{=} T_2'\}$ and $\sigma \models E \stackrel{?}{=} E'$. By induction hypothesis there exists $(\mathscr{C}_0, \mathscr{E}_0) \in \mathsf{unify}(\mathscr{C}' \cup \{T_1 \stackrel{?}{=} T_1', T_2 \stackrel{?}{=} T_2'\}, \mathscr{D})$ such that $\sigma \models \mathscr{C}_0$ and $\sigma \models \mathscr{E}_0$. Then $(\mathscr{C}_0, \mathscr{E}_0 \cup \{E \stackrel{?}{=} E'\}) \in \mathsf{unify}(\mathscr{C}, \mathscr{D})$ and $\sigma \models \mathscr{E}_0 \cup \{E \stackrel{?}{=} E'\}$.

For Item 4 we prove the case $\mathscr{C} = \mathscr{C}' \oplus \{\bullet^E(T_1 \ \mathsf{op} \ T_2) \stackrel{?}{=} \bullet^{E'}(T_1' \ \mathsf{op} \ T_2')\}$. Suppose $(\mathscr{C}_0, \mathscr{E}_0) \in \mathsf{unify}(\mathscr{C}, \mathscr{D})$ and $\rho \models \mathscr{E}_0$. Then $\mathscr{E}_0 = \mathscr{E}_0' \cup \{E \stackrel{?}{=} E'\}$. By induction hypothesis $\tau_{\mathscr{C}_0} \cup \rho \models \mathscr{C}' \cup \{T_1 \stackrel{?}{=} T_1', T_2 \stackrel{?}{=} T_2'\}$. Since $\rho \models \{E \stackrel{?}{=} E'\}$, we have that $\tau_{\mathscr{C}_0} \cup \rho \models \mathscr{C}$.

The unification algorithm is exponential on the size of the input. If we are only interested in knowing if the program is typeable or not, then the complexity could be reduced to PSpace since it would be sufficient to store just one solution at a time.

4.4 Generation of Guard Constraints

Table 3 defines an algorithm for computing the set of integer constraints needed to enforce that the types are guarded. Intuitively, the function $\mathsf{gE}(X, T)$ adds up the exponents of the bullets from the root of T to wherever X occurs in T and gC forces the resulting integer expression to be greater than 0. However, just forcing $\mathsf{gE}(X, T)$ to be greater than 0 does not work because $\mathscr{C}$ is a set of mutually recursive equations. The simplest way to track the exponents along several recursive equations seems to perform substitutions in the spirit of Bekič law [14]. Suppose $\mathscr{C} = \{X_1 = T_1, X_2 = T_2\}$. The constraint $\mathsf{gE}(X_1, T_1) > 0$ is sufficient only if X_1 does not occur in T_2. If X_1 occurs in T_2, however, we should also be able to "see the recursive occurrences of X_1" coming from the second recursive equation. For this, we define S_i and R_i similarly to the way one calculates the solution in μ-notation from a set of recursive equations (see Theorem 8.1.1 in [5]). We can omit the μ's because they are not needed.

Since the solutions of the recursive equations $X = \bullet(X \to Y)$ and $X = \bullet X \to Y$ are different and both guarded, the integer constraint to guarantee

Table 3. Generation of guard constraints

Let $\mathscr{C} = \{X_1 \stackrel{?}{=} T_1, \ldots, X_n \stackrel{?}{=} T_n\}$ be substitutional. We compute the set of inequality constraints as follows.

$$\mathsf{gC}(\mathscr{C}) = \bigcup\{0 \stackrel{?}{<} E \mid E \in \mathsf{gE}(X_i, R_i) \ \ 1 \le i \le n\}$$

where $R_1, \ldots, R_n$ are obtained by performing substitutions:

$$S_1 = T_1,\ S_2 = T_2\{X_1 \mapsto S_1\},\ \ldots,\ S_n = T_n\{X_1 \mapsto S_1\}\ldots\{X_{n-1} \mapsto S_{n-1}\}$$
$$R_n = S_n,\ R_{n-1} = S_{n-1}\{X_n \mapsto S_n\},\ \ldots,\ R_1 = S_1\{X_2 \mapsto S_2, \ldots, X_n \mapsto S_n\}$$

and gE is defined as follows:

$$\begin{aligned} \mathsf{gE}(X, X) &= \{0\} \\ \mathsf{gE}(X, \bullet^E T) &= \{E + E' \mid E' \in \mathsf{gE}(T)\} \\ \mathsf{gE}(X, T_1 \ \mathsf{op} \ T_2) &= \mathsf{gE}(X, T_1) \cup \mathsf{gE}(X, T_2) \end{aligned}$$

guardedness for $X = \bullet^N(\bullet^M X \to Y)$ should be $N + M \geq 1$ which is more general than just $M \geq 1$. It gets more complicated if we have several mutual recursive equations because the recursive variable has to be tracked through several equations. For example, consider the set

$$X_1 \stackrel{?}{=} \bullet^{N_1}(\bullet^{N_2} X_1 \to \bullet^{N_3} X_2) \quad X_2 \stackrel{?}{=} \bullet^{M_1}(\bullet^{M_2} X_2 \to \bullet^{M_3} X_3) \quad X_3 \stackrel{?}{=} \bullet^{K_1}(\bullet^{K_2} X_1 \to \bullet^{K_3} X_3)$$

then, the set $\mathscr{E}$ of integer constraints to enforce that the types are guarded is:

$$\{N_1 + N_2 \geq 1, N_1 + N_3 + M_1 + M_3 + K_1 + K_2 \geq 1, M_1 + M_2 \geq 1, K_1 + K_3 \geq 1\}$$

Theorem 5 (Correctness of the Guard Constraint Generation). *Let* $\mathscr{C} = \{X_1 \stackrel{?}{=} T_1, \ldots, X_n \stackrel{?}{=} T_n\}$ *be substitutive and* $\tau_{\mathscr{C}} = \{X_1 \mapsto S_1, \ldots, X_n \mapsto S_n\}$ *and* $S_i\rho$ *be grounded. Then,* $S_i\rho$ *is guarded for all* $1 \leq i \leq n$ *iff* $\rho \models \mathsf{gC}(\mathscr{C})$.

Proof. Suppose $\rho \not\models \mathsf{gC}(\mathscr{C})$. Then, there is $E \in \mathsf{gE}(X_i, R_i)$ such that $\rho(E) = 0$ then the path from the root of $R_i\rho$ to X_i has no bullets. Since $S_i\rho = T_i(\rho \circ \tau_{\mathscr{C}}) = R_i(\rho \circ \tau_{\mathscr{C}'})$ where $\tau_{\mathscr{C}'} = \{X_1 \mapsto R_1, \ldots, X_n \mapsto R_n\}$, we have that there exists an infinite path in $S_i\rho$ that has no bullets at all. For the converse, suppose $\rho \models \mathsf{gC}(\mathscr{C})$. Then, an infinite path in $S_i\rho$ is an infinite path in $R_i(\rho \circ \tau_{\mathscr{C}'})$ and this path has an infinite number of bullets because the path from the root of $R_i\rho$ to X_i has a number $n = E\rho > 0$ of bullets.

4.5 Type Inference Algorithm

The *type inference algorithm* defined in Table 4 returns a finite set of meta-types that cover all the possible types of a closed expression e, i.e. any type of e is an instantiation of one of those meta-types. Item 2a checks that the type is different from $\bullet^\infty$. Here, we are using the fact that $t \neq \bullet^\infty$ if and only if t is either $\bullet^n X$ or $\bullet^n(t_1 \to t_2)$ or $\bullet^n(t_1 \times t_2)$. In order to check that $\mathscr{E} \cup \mathsf{gC}(\tau_{\mathscr{C}})$ has a solution of non-negative integers in Item 2b, we can use any algorithm for linear integer programming [18]. It is easy to modify this algorithm to give a set of minimal solutions by minimizing the sum of all integer variables. For instance, if the solution is a meta-type T satisfying the recursive equation $T = \bullet^{N_1}(\bullet^{N_2} T \to S)$ with $N_1 + N_2 \geq 1$ then we have two minimal solutions $N_1 = 1$, $N_2 = 0$ and $N_1 = 0$, $N_2 = 1$.

As an example, consider $\lambda x.x$. Then $\mathsf{infer}(\lambda x.x)$ yields only one solution $\bullet^N(X \to \bullet^M X)$ which it is actually its most general type.

Consider now $\lambda x.xx$. Then $\mathsf{infer}(\lambda x.xx)$ gives a set of two meta-types. The first meta-type is $\bullet^{N_1}(X_1 \to \bullet^{N_2 - N_1} X_4)$ where X_1 should satisfy the recursive equation

$$X_1 \stackrel{?}{=} \bullet^{N_2 - (N_1 + N_3)}(\bullet^{N_1 + N_4 - N_2} X_1 \to X_4)$$

and N_1, N_2, N_3, N_4 should all be non-negative and satisfy $N_1 + N_4 \geq N_2 \geq N_1 + N_3$ and $N_4 - N_3 \geq 1$. The second meta-type is $\bullet^{N_1}(\bullet^{N_2 - (N_1 + N_4)} X_3 \to \bullet^{N_2 - N_1} X_4)$ where X_3 should satisfy the recursive equation

$$X_3 \stackrel{?}{=} \bullet^{N_4 - N_3}(X_3 \to X_4)$$

Table 4. Type inference algorithm

Let e be a closed expression. The type inference algorithm $\mathsf{infer}(e)$ proceeds as follows.

1. It first calculates the meta-type T_0 and the set $\mathscr{C}_0$ of meta-type constraints such that $\emptyset \vdash e : T_0 \mid \mathscr{C}_0$ using the typing rules of Table 1.
2. Then $\mathsf{infer}(e)$ returns the set of pairs $(T_0\tau_{\mathscr{C}}, \mathscr{E})$ such that $(\mathscr{C}, \mathscr{E}) \in \mathsf{unify}(\mathscr{C}_0, \mathscr{C}_0)$ and the following two conditions hold:
   (a) $T_0\tau_{\mathscr{C}}$ is either $\bullet^E X$ or $\bullet^E(T_1 \to T_2)$ or $\bullet^E(T_1 \times T_2)$ and
   (b) the set $\mathscr{E} \cup \mathsf{gC}(\tau_{\mathscr{C}})$ has a solution of non-negative integers where gC is the set of constraints computed as in Table 3.

and N_1, N_2, N_3, N_4 should all be non-negative and satisfy $N_2 > N_1 + N_4$ and $N_4 \geq N_3$ and $N_4 - N_3 \geq 1$. These two meta-types "cover all solutions" in the sense that any type of $\lambda x.xx$ is an instantiation of one of these two meta-types.

Theorem 6 (Correctness of Type Inference).

1. $\mathsf{infer}(e)$ *gives a finite set of pairs* $(T, \mathscr{E})$ *such that there exists at least one ground substitution* ρ *such that* $\rho \models \mathscr{E}$.
2. *If* $(T, \mathscr{E}) \in \mathsf{infer}(e)$ *then* $\vdash e : t$ *and* $T\rho = t \neq \bullet^\infty$ *for all ground substitutions* $\rho \models \mathscr{E}$.
3. *If* $\vdash e : t$ *and* $t \neq \bullet^\infty$ *then there exists* $(T, \mathscr{E}) \in \mathsf{infer}(e)$ *such that* $T\sigma = t$ *and* $\sigma \models \mathscr{E}$.

Proof. This follows from Theorems 3, 4 and 5.

Type checking for $\lambda^\bullet_\to$ (given an an expression e and a type t check if $\vdash e : t$) can easily be solved by inferring the (finite) set of meta-types for e and checking whether one of these meta-types unifies with t.

We could try to define an alternative type inference algorithm for $\lambda^\bullet_\to$ by re-using the one for $\lambda\mu$ [4,5]. However, this option does not make the problem simpler. Consider

$$\mathtt{skipRep}\ xs = \langle \mathtt{fst}\ xs, \langle \mathtt{fst}\ xs, \mathtt{skipRep}\ (\mathtt{snd}\ (\mathtt{snd}\ xs))\rangle\rangle$$

This function has type $\mathtt{Str}_{\mathtt{Nat}} \to \mathtt{SStr2}_{\mathtt{Nat}}$ where $\mathtt{SStr2}_{\mathtt{Nat}} = \mathtt{Nat} \times \mathtt{Nat} \times \bullet\bullet\ \mathtt{SStr2}_{\mathtt{Nat}}$. The type of this function without bullets is $\mathtt{Str}'_{\mathtt{Nat}} \to \mathtt{Str}'_{\mathtt{Nat}}$. How can we decorate $\mathtt{Str}'_{\mathtt{Nat}} \to \mathtt{Str}'_{\mathtt{Nat}}$ in order to obtain $\mathtt{Str}_{\mathtt{Nat}} \to \mathtt{SStr2}_{\mathtt{Nat}}$? Since a recursive type can be seen as an infinite tree with a finite number k of sub-trees, we could decorate each sub-tree with a certain number n_k of bullets. However, since $\mathtt{Str}_{\mathtt{Nat}}$ has only one sub-tree, it would mean that all sub-trees are decorated with the same number of bullets. In order to obtain $\mathtt{SStr2}_{\mathtt{Nat}}$, the domain $\mathtt{Str}'_{\mathtt{Nat}}$ of the function can be decorated by putting 1 bullet in all its subtrees. However, in order to obtain $\mathtt{SStr2}_{\mathtt{Nat}}$, the range $\mathtt{Str}'_{\mathtt{Nat}}$ of the function needs to be decorated by putting 0 bullets in some subtrees and 1 in others.

5 Related Work

This paper is an improvement over past typed lambda calculi with a temporal modal operator in two respects. Firstly, we do not need any subtyping relation as in [17] and secondly programs are not cluttered with constructs for the introduction and elimination of individuals of type $\bullet$ as in [1,3,6,7,15,16,23]. The type system of [3] is designed having that denotational semantics in mind and it is syntactically more involved. Section 3 shows that our type system being syntactically simpler (and not designed having the semantics in mind) still admits the same semantics.

If we restrict to the recursive types $\mathtt{Str}_t = t \times \bullet \mathtt{Str}_t$ for infinite lists then, $\lambda^{\bullet}_{\rightarrow}$ is essentially the same as the type system of [15]. If only recursive types for lists are available, one cannot create other recursive types such as trees but having only one bullet also limits the amount of functions on streams we can type, e.g. `skip` is not typable in the type systems of [15,23].

Another type-based approach for ensuring productivity are sized types [13]. Type systems using size types do not always have neat properties: strong normalisation is gained by contracting the fixed point operator inside a case and they lack the property of subject reduction [22]. Another disadvantage of size types is that they do not include negative occurrences of the recursion variable [13] which are useful for some applications [26].

The proof assistant Coq does not ensure productivity through typing but by means of a syntactic guardedness condition (the recursive calls should be guarded by constructors) [8,11] which is somewhat restrictive since it rules out some interesting functions [7,23].

A sound but not complete type inference algorithm for Nakano's type system is presented in [21]. This means that the expressions typable by the algorithm are also typable in Nakano's system but the converse is not true. Though this algorithm is tractable, it is not clear to which type system it corresponds.

6 Conclusions and Future Work

The typeability problem (finding out if the program is typeable or not) is trivial in $\lambda\mu$ because all expressions are typeable using $\mu X.X \rightarrow X$. In $\lambda^{\bullet}_{\rightarrow}$, this problem turns out to be interesting because it gives us a way of filtering non-normalising programs when the type is not $\bullet^{\infty}$. It is also challenging because it involves the generation of integer constraints. The type inference algorithm presented in this paper does not give a unique principal type but a finite set of "principal" types. Since this (finite) set of types covers *all* possible types, it is possible to have modularity. Moreover, this algorithm can be easily extended to infer types for the processes of [24] since an initial process is the parallel composition of the main thread $x \Leftarrow e$ with an arbitrary number of $(\mathtt{server}\quad a\ e_i)$ where e and e_i are closed expressions typable in $\lambda^{\bullet}_{\rightarrow}$ extended with `IO` and session types. As we mentioned at the end of Sect. 4.3, the typeability problem in $\lambda^{\bullet}_{\rightarrow}$ can be solved in PSpace but it still remains to see if this problem is PSpace-hard. In any

case, it is important to refine this algorithm and find optimization techniques (heuristics, use of concurrency, etc.) to make it practical.

It will be interesting to investigate the interaction of this variant of the modal operator with dependent types. As observed in Sect. 2, our type system is not closed under η-reduction. We leave the challenge of attaining a normalising and decidable type system closed under $\beta\eta$-reduction for future research.

References

1. Atkey, R., McBride, C.: Productive coprogramming with guarded recursion. In: Morrisett, G., Uustalu, T. (eds.) Proceedings of ICFP 2013, pp. 197–208. ACM (2013)
2. Birkedal, L., Møgelberg, R.E., Schwinghammer, J., Støvring, K.: First steps in synthetic guarded domain theory: step-indexing in the topos of trees. Log. Methods Comput. Sci. **8**(4), 1–45 (2012)
3. Bizjak, A., Grathwohl, H.B., Clouston, R., Møgelberg, R.E., Birkedal, L.: Guarded dependent type theory with coinductive types. In: Jacobs, B., Löding, C. (eds.) FoSSaCS 2016. LNCS, vol. 9634, pp. 20–35. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49630-5_2
4. Cardone, F., Coppo, M.: Type inference with recursive types: Syntax and semantics. Inf. Comput. **92**(1), 48–80 (1991)
5. Cardone, F., Coppo, M.: Recursive types. In: Barendregt, H., Dekkers, W., Statman, R. (eds.) Lambda Calculus with Types, Perspectives in Logic, pp. 377–576. Cambridge (2013)
6. Cave, A., Ferreira, F., Panangaden, P., Pientka, B.: Fair reactive programming. In: Jagannathan, S., Sewell, P. (eds.) Proceedings of POPL 2014, pp. 361–372. ACM Press (2014)
7. Clouston, R., Bizjak, A., Grathwohl, H.B., Birkedal, L.: Programming and reasoning with guarded recursion for coinductive types. In: Pitts, A. (ed.) FoSSaCS 2015. LNCS, vol. 9034, pp. 407–421. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46678-0_26
8. Coquand, T.: Infinite objects in type theory. In: Barendregt, H., Nipkow, T. (eds.) TYPES 1993. LNCS, vol. 806, pp. 62–78. Springer, Heidelberg (1994). doi:10.1007/3-540-58085-9_72
9. Courcelle, B.: Fundamental properties of infinite trees. Theoret. Comput. Sci. **25**, 95–169 (1983)
10. Gapeyev, V., Levin, M.Y., Pierce, B.C.: Recursive subtyping revealed. J. Funct. Program. **12**(6), 511–548 (2002)
11. Giménez, E., Casterán, P.: A tutorial on [co-]inductive types in coq. Technical report, Inria (1998)
12. Hughes, J.: Why functional programming matters. Comput. J. **32**(2), 98–107 (1989)
13. Hughes, J., Pareto, L., Sabry, A.: Proving the correctness of reactive systems using sized types. In: Proceedings of POPL 1996, pp. 410–423 (1996)
14. Jones, C. (ed.): Programming Languages and Their Definition - Hans Bekic (1936–1982). LNCS, vol. 177. Springer, Heidelberg (1984)
15. Krishnaswami, N.R., Benton, N.: Ultrametric semantics of reactive programs. In: Grohe, M. (ed.) Proceedings of LICS 2011, pp. 257–266. IEEE (2011)

16. Krishnaswami, N.R., Benton, N., Hoffmann, J.: Higher-order functional reactive programming in bounded space. In: Proceedings of POPL 2012, pp. 45–58. ACM Press (2012)
17. Nakano, H.: A modality for recursion. In: Abadi M. (ed.) Proceedings of LICS 2000, pp. 255–266. IEEE (2000)
18. Papadimitriou, C.H., Steiglitz, K.: Combinatorial Optimization: Algorithms and Complexity. Dover Publications, Mineola (1998)
19. Pierce, B.C.: Types and Programming Languages. MIT Press, Cambridge (2002)
20. Pottier, F., Rémy, D.: The essence of ML type inference. In: Pierce, B.C. (ed.) Advanced Topics in Types and Programming Languages, pp. 389–489. MIT Press (2005)
21. Rowe, R.N.S.: Semantic Types for Class-based Objects. Ph.D. thesis, Imperial College London (2012)
22. Sacchini, J.L.: Type-based productivity of stream definitions in the calculus of constructions. In: Procedings of LICS 2013, pp. 233–242 (2013)
23. Severi, P., de Vries, F.-J.: Pure type systems with corecursion on streams: from finite to infinitary normalisation. In: Thiemann, P., Findler, R.B. (eds.) Proceedings of ICFP 2012, pp. 141–152. ACM Press (2012)
24. Severi, P., Padovani, L., Tuosto, E., Dezani-Ciancaglini, M.: On sessions and infinite data. In: Lluch Lafuente, A., Proença, J. (eds.) COORDINATION 2016. LNCS, vol. 9686, pp. 245–261. Springer, Cham (2016). doi:10.1007/978-3-319-39519-7_15
25. Severi, P., Padovani, L., Tuosto, E., Dezani-Ciancaglini, M.: On sessions and infinite data. CoRR, abs/1610.06362 (2016)
26. Svendsen, K., Birkedal, L.: Impredicative concurrent abstract predicates. In: Shao, Z. (ed.) ESOP 2014. LNCS, vol. 8410, pp. 149–168. Springer, Heidelberg (2014). doi:10.1007/978-3-642-54833-8_9

Unifying Guarded and Unguarded Iteration

Sergey Goncharov[1(✉)], Lutz Schröder[1], Christoph Rauch[1], and Maciej Piróg[2]

[1] Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, Germany
sergey.goncharov@fau.de
[2] Department of Computer Science, KU Leuven, Leuven, Belgium

Abstract. Models of iterated computation, such as (completely) iterative monads, often depend on a notion of guardedness, which guarantees unique solvability of recursive equations and requires roughly that recursive calls happen only under certain guarding operations. On the other hand, many models of iteration do admit unguarded iteration. Solutions are then no longer unique, and in general not even determined as least or greatest fixpoints, being instead governed by quasi-equational axioms. Monads that support unguarded iteration in this sense are called (complete) Elgot monads. Here, we propose to equip monads with an abstract notion of guardedness and then require solvability of abstractly guarded recursive equations; examples of such *abstractly guarded pre-iterative monads* include both iterative monads and Elgot monads, the latter by deeming any recursive definition to be abstractly guarded. Our main result is then that Elgot monads are precisely the iteration-congruent retracts of abstractly guarded *iterative* monads, the latter being defined as admitting *unique* solutions of abstractly guarded recursive equations; in other words, models of unguarded iteration come about by quotienting models of guarded iteration.

1 Introduction

In recursion theory, notions of guardedness traditionally play a central role. Guardedness typically means that recursive calls must be in the scope of certain guarding operations, a condition aimed, among other things, at ensuring progress. The paradigmatic case are recursive definitions in process algebra, which are usually called guarded if recursive calls occur only under action prefixing [6]. A more abstract example are completely iterative theories [11] and monads [19], where, in the latter setting, a recursive definition is guarded if it factors through a given ideal of the monad. Guarded recursive definitions typically have unique solutions; e.g. the unique solution of the guarded recursive definition

$$X = a.\, X$$

is the process that keeps performing the action a.

For unguarded recursive definitions, the picture is, of course, different. E.g. to obtain the denotational semantics of an unproductive while loop $\mathsf{while\ true\ do\ skip}$ characterized by circular operational behavior

$$\mathsf{while\ true\ do\ skip} \to \mathsf{skip}; \mathsf{while\ true\ do\ skip} \to \mathsf{while\ true\ do\ skip}$$

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 517–533, 2017.
DOI: 10.1007/978-3-662-54458-7_30

one will select one of many solutions of this trivial equation, e.g. the least solution in a domain-theoretic semantics.

Sometimes, however, one has a selection among non-unique solutions of unguarded recursive equations that is *not* determined order-theoretically, i.e. by picking least or greatest fixpoints. One example arises from *coinductive resumptions* [16,25,26]. In the paradigm of monad-based encapsulation of side-effects [21], coinductive resumptions over a base effect encapsulated by a monad T form a *coinductive resumption transform* monad T^ν given by

$$T^\nu X = \nu\gamma.\, T(X + \gamma) \tag{1}$$

– that is, a computation over X performs a step with effects from T, and then returns either a value from X or a resumption that, when resumed, proceeds similarly, possibly ad infinitum. We generally restrict to monads T for which (1) exists for all X (although many of our results do not depend on this assumption). Functors (or monads) T for which this holds are called *iteratable* [1]. Most computationally relevant monads are iteratable (notable exceptions in the category of sets are the powerset monad and the continuation monad). Notice that one has a natural *delay map* $T^\nu X \to T^\nu X$ that converts a computation into a resumption, i.e. prefixes it with a delay step. In fact, for $T = \mathsf{id}$, T^ν is precisely Capretta's *partiality monad* [8], also called the *delay monad*. It is not in general possible to equip $T^\nu X$ with a domain structure that would allow for selecting least or greatest solutions of unguarded recursive definitions over T^ν. However, one *can* select solutions in a coherent way, that is, such that a range of natural quasi-equational axioms is satisfied, making T^ν into a (complete) *Elgot monad* [2,15].

In the current work we aim to unify theories of guarded and unguarded iteration. To this end, we introduce a notion of *abstractly guarded monads*, that is, monads T equipped with a distinguished class of *abstractly guarded* equation morphisms satisfying some natural closure properties (Sect. 3). The notion of abstract guardedness can be instantiated in various ways, e.g. with the class of immediately terminating 'recursive' definitions, with the class of guarded morphisms in a completely iterative monad, or with the class of all equation morphisms. We call an abstractly guarded monad *pre-iterative* if all abstractly guarded equation morphisms have a solution, and *iterative* if these solutions are unique. Then completely iterative monads are iterative abstractly guarded monads in this sense, and (complete) Elgot monads are pre-iterative, where we deem every equation morphism to be abstractly guarded in the latter case.

The quasi-equational axioms of Elgot monads are easily seen to be satisfied when fixpoints are unique, i.e. in iterative abstractly guarded monads, and moreover stable under iteration-congruent retractions in a fairly obvious sense. Our first main result (Sect. 5, Theorem 22) states that the converse holds as well, i.e. *a monad T is a complete Elgot monad iff T is an iteration-congruent retract of an iterative abstractly guarded monad* – specifically of T^ν as in (1). As a slogan,

> monad-based models of unguarded recursion arise by quotienting models of guarded recursion.

Our second main result (Theorem 26) is an algebraic characterization of complete Elgot monads: We show that the construction $(-)^\nu$ mapping a monad T to T^ν as in (1) is a monad on the category of monads (modulo existence of T^ν), and *complete Elgot monads are precisely those* $(-)^\nu$*-algebras* T *that cancel the delay map on* T^ν, i.e. interpret the delay operation as identity.

As an illustration of these results, we show (Sect. 6) that sandwiching a complete Elgot monad between adjoint functors again yields a complete Elgot monad, in analogy to a corresponding result for completely iterative monads [26]. Specifically, we prove a sandwich theorem for iterative abstractly guarded monads and transfer it to complete Elgot monads using our first main result. For illustration, we then relate iteration in ultrametric spaces using Escardó's metric lifting monad [12] to iteration in pointed cpo's, by noting that the corresponding monads on sets obtained using our sandwich theorems are related by an iteration-congruent retraction in the sense of our first main result.

2 Preliminaries

We work in a category $\mathbf{C}$ with finite coproducts. We fix the notation $\mathsf{in}_i : X_i \to X_1 + \ldots + X_n$ for the i-th injection. A morphism $\sigma : Y \to X$ is a *summand* of X, which we denote $\sigma : Y \rightarrowtail X$, if there is $\sigma' : X' \to X$ such that X is a coproduct of Y and X' with σ and σ' being coproduct injections. The morphism σ' is called a *(coproduct) complement* of σ and by definition is also a summand. We are not assuming that $\mathbf{C}$ is *extensive*, and coproduct complements are not in general uniquely determined.

A *monad* $\mathbb{T}$ over $\mathbf{C}$ can be given in a form of a *Kleisli triple* $(T, \eta, -^\star)$ where T is an endomap over the objects $|\mathbf{C}|$ of $\mathbf{C}$, the unit η is a family of morphisms $(\eta_X : X \to TX)_{X \in |\mathbf{C}|}$, *Kleisli lifting* $(-)^\star$ is a family of maps : $\mathsf{Hom}(X, TY) \to \mathsf{Hom}(TX, TY)$, and the *monad laws* are satisfied:

$$\eta^\star = \mathsf{id}, \qquad f^\star \eta = f, \qquad (f^\star g)^\star = f^\star g^\star.$$

The standard (equivalent) categorical definition [18] of $\mathbb{T}$ as an endofunctor with natural transformation *unit* $\eta : \mathrm{Id} \to T$ and *multiplication* $\mu : TT \to T$ can be recovered by taking $Tf = (\eta\, f)^\star$, $\mu = \mathsf{id}^\star$. (We adopt the convention that monads and their functor parts are denoted by the same letter, with the former in blackboard bold.) We call morphisms $X \to TY$ *Kleisli morphisms* and view them as a high level abstraction of sequential programs with $\mathbb{T}$ encapsulating the underlying computational effect as proposed by Moggi [22], with X representing the input type and Y the output type. A more traditional use of monads in semantics is due to Lawvere [17], who identified finitary monads on **Set** with *algebraic theories*, hence objects TX can be viewed as sets of terms of the theory over free variables from X, the unit as the operation of casting a variable to a term, and Kleisli composition as substitution. We informally refer to this use of monads as *algebraic monads*.

A(n F*-)coalgebra* is a pair $(X, f : X \to FX)$ where $X \in |\mathbf{C}|$ and $F : \mathbf{C} \to \mathbf{C}$ is an endofunctor. Coalgebras form a category, with morphisms $(X, f) \to (Y, g)$

being $\mathbf{C}$-morphisms $h : X \to Y$ such that $(Fh)\, f = gh$. A final object of this category is called a *final coalgebra*, and we denote it by $(\nu F, \mathsf{out} : \nu F \to F\nu F)$ if it exists. For brevity, *we will be cavalier about existence of final coalgebras and silently assume they exist when we need them*; that is, we hide sanity conditions on the involved functors, such as accessibility. By definition, νF comes with *coiteration* as a definition principle (dual to the iteration principle for algebras): given a coalgebra $(X, f : X \to FX)$ there is a unique morphism $(\mathsf{coit}\ f) : X \to \nu F$ such that

$$\mathsf{out}\,(\mathsf{coit}\ f) = F(\mathsf{coit}\ f)\, f.$$

This implies that out is an isomorphism (*Lambek's lemma*) and that $\mathsf{coit}\ \mathsf{out} = \mathsf{id}$ (see [29] for more details about coalgebras for coiteration).

We generally drop sub- and superscripts, e.g. on natural transformations, whenever this improves readability.

3 Guarded Monads

The notion of guardedness is paramount in process algebra: typically one considers systems of mutually recursive process definitions of the form $x_i = t_i$, and a variable x_i is said to be guarded in t_j if it occurs in t_j only in subterms of the form $a.s$ where $a.(\text{–})$ is action prefixing. A standard categorical approach is to replace the set of terms over variables X by an object TX where $\mathbb{T}$ is a monad. We then can model separate variables by partitioning X into a sum $X_1 + \ldots + X_n$ and thus talk about guardedness of a morphism $f : X \to T(X_1 + \ldots + X_n)$ in any X_i, meaning that every variable from X_i is guarded in f. Since Kleisli morphisms can be thought of as abstract programs we can therefore speak about guardedness of a program in a certain portion of the output type, e.g. $X_i \hookrightarrow X_1 + \ldots + X_n$. One way to capture guardedness categorically is to identify the operations of $\mathbb{T}$ that serve as guards by distinguishing a suitable subobject of $\mathbb{T}$; e.g. the definition of completely iterative monad [19] follows this approach. For our purposes, we require a yet more general notion where we just distinguish some Kleisli morphisms as being guarded in certain output variables. This is formalized as follows.

Definition 1 ((Abstractly) guarded monad). We call a monad $\mathbb{T}$ *(abstractly) guarded* if it is equipped with a notion of (abstract) guardedness, i.e. with a relation between morphisms $f : X \to TY$ and summands of Y (by putting the word 'abstract' in brackets we mean that we will often omit it later). We call $f : X \to TY$ *(abstractly) σ-guarded* if f and $\sigma : Y' \hookrightarrow Y$ are in this relation, and then write $f : X \to_\sigma TY$. Abstract guardedness is required to be closed under the rules in Fig. 1. In rule **(wkn)**, σ and θ are composable summands.

Given guarded monads $\mathbb{T}$, $\mathbb{S}$, a monad morphism $\alpha : T \to S$ is *(abstractly) guarded* if whenever $f : X \to_\sigma TY$ then $\alpha f : X \to_\sigma SY$.

Intuitively, **(trv)** says that if a program does not output anything via a summand of the output type then it is guarded in that summand. Rule **(wkn)** is a

$$\textbf{(trv)}\quad \frac{f : X \to TY}{(T\,\mathsf{in}_1)\,f : X \to_{\mathsf{in}_2} T(Y+Z)} \qquad \textbf{(wkn)}\quad \frac{f : X \to_\sigma TY}{f : X \to_{\sigma\theta} TY}$$

$$\textbf{(cmp)}\quad \frac{f : X \to_{\mathsf{in}_2} T(Y+Z) \qquad g : Y \to_\sigma TV \qquad h : Z \to TV}{[g,h]^\star f : X \to_\sigma TV}$$

$$\textbf{(sum)}\quad \frac{f : X \to_\sigma TZ \qquad g : Y \to_\sigma TZ}{[f,g] : X + Y \to_\sigma TZ}$$

Fig. 1. Axioms of guardedness.

weakening principle: if a program is guarded in some summand then it is guarded in any subsummand of that summand. Rule **(cmp)** asserts that guardedness is preserved by composition: if the unguarded part of the output of a program is postcomposed with a σ-guarded program then the result is σ-guarded, no matter how the guarded part is transformed. Finally, rule **(sum)** says that putting two guarded equation systems side by side again produces a guarded system. The rules are designed so as to enable a reformulation of the classical laws of iteration w.r.t. abstract guardedness, as we shall see in Sect. 5.

We write $f : X \to_{i_1,\ldots,i_k} T(X_1 + \ldots + X_n)$ as a shorthand for $f : X \to_\sigma T(X_1 + \ldots + X_n)$ with $\sigma = [\mathsf{in}_{i_1}, \ldots, \mathsf{in}_{i_k}] : X_{i_1} + \ldots + X_{i_k} \hookrightarrow X_1 + \ldots + X_n$. More generally, we sometimes need to refer to components of some X_{i_j}. This amounts to replacing the corresponding i_j with a sequence of pairs $i_j n_{j,m}$, and in_{i_j} with $\mathsf{in}_{i_j}[\mathsf{in}_{n_{j,1}}, \ldots, \mathsf{in}_{n_{j,k_j}}]$, so, e.g. we write $f : X \to_{12,2} T((Y+Z)+Z)$ to mean that f is $[\mathsf{in}_1\,\mathsf{in}_2, \mathsf{in}_2]$-guarded. Where coproducts $Y + Z$ etc. appear in the rules, we mean any coproduct, not just some selected coproduct. We defined the notion of guardedness as a certain relation over Kleisli morphisms and summands. Clearly, the largest such relation is the one declaring all Kleisli morphisms to be σ-guarded for all σ. We call monads equipped with this notion of guardedness *totally guarded*. It turns out that for every monad we also have a least guardedness relation.

Definition 2. Let $\mathbb{T}$ be a monad. A morphism $f : X \to TY$ is *trivially σ-guarded* for $\sigma : Z \hookrightarrow Y$ if f factors through $T\sigma'$ for a coproduct complement σ' of σ.

Proposition 3. *Let $\mathbb{T}$ be a monad. Then taking the abstractly guarded morphisms to be the trivially guarded morphisms is the least guardedness relation making $\mathbb{T}$ into a guarded monad.*

We call a guarded monad *trivially guarded* if all its abstractly guarded morphisms are trivially guarded. As we see, the notion of abstract guardedness can vary on a large spectrum from *trivial guardedness* to *total guardedness*, thus somewhat detaching abstract guardedness from the original intuition. It is for this reason that we introduced the qualifier *abstract* into the terminology; for brevity, we will omit this qualifier in the sequel in contexts where no confusion is likely, speaking only of guarded monads, guarded morphisms etc.

Remark 4. Although by **(wkn)**, $f : X \to_{1,2} T(Y + Z)$ implies both $f : X \to_1 T(Y + Z)$ and $f : X \to_2 T(Y + Z)$, the converse is not required to be true, and in fact can fail even for trivial guardedness. This is witnessed by the following simple counterexample. Let $\mathbb{T}$ be the algebraic monad given by taking TX to be the free commutative semigroup over X satisfying the additional law $x + y = x$. Now the term $x + y \in T(X + Y)$ (seen as a morphism $1 \to T(X + Y)$) with $x \in X$ and $y \in Y$ is both in_1-guarded and in_2-guarded, being equivalent both to y and to x. But it is not id-guarded, because it does not factor through $T\emptyset = \emptyset$.

As usual, guardedness serves to identify systems of equations that admit solutions according to some global principle:

Definition 5 (Guarded (pre-)iterative monad). Given $f : X \to_2 T(Y+X)$, we say that $f^\dagger : X \to TY$ is a *solution* of f if $f^\dagger$ satisfies the *fixpoint identity* $f^\dagger = [\eta, f^\dagger]^\star f$. A guarded monad $\mathbb{T}$ is *guarded pre-iterative* if it is equipped with an *iteration operator* that assigns to every in_2-guarded morphism $f : X \to_2 T(Y + X)$ a solution $f^\dagger$ of f. If every such f has a unique solution, we call $\mathbb{T}$ *guarded iterative*.

We can readily check that the iteration operator preserves guardedness.

Proposition 6. *Let $\mathbb{T}$ be a guarded pre-iterative monad, let $\sigma : Y' \hookrightarrow Y$, and let $f : X \to_{\sigma+\mathsf{id}} T(Y + X)$. Then $f^\dagger : X \to_\sigma TY$.*

In trivially guarded monads, there is effectively nothing to iterate, so we have

Proposition 7. *Every trivially guarded monad is guarded iterative.*

Guardedness in Completely Iterative Monads. One instance of our notion of abstract guardedness is found in completely iterative monads [19], which are based on idealised monads. To make this precise, we need to recall some definitions. First, a *module* over a monad $\mathbb{T}$ on $\mathbf{C}$ is a pair $(M, -^\circ)$, where M is an endomap over the objects of $\mathbf{C}$, while the lifting $-^\circ$ is a map $\mathsf{Hom}(X, TY) \to \mathsf{Hom}(MX, MY)$ such that the following laws are satisfied:

$$\eta^\circ = \mathsf{id}, \qquad\qquad g^\circ f^\circ = (g^\star f)^\circ.$$

Note that M extends to an endofunctor by taking $Mf = (\eta f)^\circ$. Next, a *module-to-monad morphism* is a natural transformation $\xi : M \to T$ that satisfies $\xi f^\circ = f^\star \xi$. We call the triple $(\mathbb{T}, M, -^\circ, \xi)$ an *idealised monad*; when no confusion is likely, we refer to these data just as $\mathbb{T}$. Following [19], we can then define guardedness as follows:

Definition 8. Given an idealised monad $\mathbb{T}$ as above, a morphism $f : X \to T(Y + X)$ is *guarded* if it factors via $[\eta\,\mathsf{in}_1, \xi] : Y + M(Y + X) \to T(Y + X)$. Then, $\mathbb{T}$ is a *completely iterative monad* if every such guarded f has a unique solution.

It turns out that this notion of guardedness is not an instance of our notion of abstract guardedness. Fortunately, we can fix this by noticing that completely iterative monads actually support iteration for a wider class of morphisms:

Definition 9. Let $(\mathbb{T}, M, -^\circ, \xi)$ be an idealised monad. Given $\sigma : Z \hookrightarrow Y$, we say that a morphism $f : X \to TY$ is *weakly σ-guarded* if it factors through $[\eta\sigma', \xi]^\star : T(Y' + MY) \to TY$ for a complement $\sigma' : Y' \hookrightarrow Y$ of σ.

Since a morphism that factors as $[\eta\,\mathsf{in}_1, \xi]f$ can be rewritten as $[\eta\,\mathsf{in}_1, \xi]^\star \eta f$, every guarded morphism in an idealised monad is also weakly guarded.

Theorem 10. *Let $(\mathbb{T}, M, -^\circ, \xi)$ be an idealised monad. Then the following hold.*

1. *$\mathbb{T}$ becomes abstractly guarded when equipped with weak guardedness as the notion of abstract guardedness.*
2. *If $\mathbb{T}$ is completely iterative, then every weakly in_2-guarded morphism $f : X \to T(Y + X)$ has a unique solution.*

That is, completely iterative monads are abstractly guarded iterative monads w.r.t. weak guardedness.

4 Parametrizing Guardedness

Uustalu [28] defines a *parametrized monad* to be a functor from a category $\mathbf{C}$ to the category of monads over $\mathbf{C}$. We need a minor adaptation of this notion where we allow parameters from a different category than $\mathbf{C}$, and simultaneously introduce a guarded version of parametrized monads:

Definition 11 (Parametrized guarded monad). A *parametrized (guarded) monad* is a functor from a category $\mathbf{D}$ to the category of (guarded) monads and (guarded) monad morphisms over $\mathbf{C}$. Alternatively (by uncurrying), it is a bifunctor $\# : \mathbf{C} \times \mathbf{D} \to \mathbf{C}$ such that for any $X \in |\mathbf{D}|$, $- \# X : \mathbf{C} \to \mathbf{C}$ is a (guarded) monad, and for every $f : X \to Y$, $\mathsf{id} \# f : Z \# X \to Z \# Y$ is the Z-component of a (guarded) monad morphism $- \# f : - \# X \to - \# Y$.

A *parametrized (guarded) monad morphism* between guarded monads qua functors into the category of monads over $\mathbf{C}$ is a natural transformation that is componentwise a monad morphism. In uncurried notation, given parametrized monads $\#, \hat{\#} : \mathbf{C} \times \mathbf{D} \to \mathbf{C}$ a natural transformation $\alpha : \# \to \hat{\#}$ is a parametrized (guarded) monad morphism if for each $X \in |\mathbf{D}|$, $\alpha_{-,X} : - \# X \to - \hat{\#} X$ is a (guarded) monad morphism.

Guardedness of the monad morphisms $- \# f$ means explicitly that $g : Z \to_\sigma V \# X$ implies $(\mathsf{id} \# f)\, g : Z \to_\sigma V \# Y$.

Example 12. For the purposes of the present work, the most important example (taken from [28]) is $\# = T(- + \Sigma -) : \mathbf{C} \times \mathbf{C} \to \mathbf{C}$ where $\mathbb{T}$ is a (non-parametrized) monad on $\mathbf{C}$ and Σ is an endofunctor on $\mathbf{C}$. Informally, $\mathbb{T}$ captures a computational effect, e.g. nondeterminism for T being powerset, and Σ

captures a signature of actions, e.g. $\Sigma X = A \times X$, as in process algebra. Specifically, taking $A = 1$ we obtain $X \mathbin{\#} Y = T(X + Y)$; in this case, we have only one guard, which can be interpreted as a delay. The second argument of $\#$ can thus be thought of as designated for guarded recursion.

Incidentally, our modification of parametrized monads also covers Atkey's parametrized monads [5], which are certain functors $\mathbf{S} \times \mathbf{S}^{op} \times \mathbf{C} \to \mathbf{C}$ forming a monad in the third argument. The first and the second arguments serve, e.g., to parametrize the computational effect of interest with initial and final states of different types.

Theorem 13. *Let $\# : \mathbf{C} \times (\mathbf{C} \times \mathbf{D}) \to \mathbf{C}$ be a parametrized monad, with unit η and Kleisli lifting $(-)^\star$. Then*

$$X \mathbin{\#^\nu} Y = \nu\gamma.\, X \mathbin{\#} (\gamma, Y)$$

defines a parametrized monad $\#^\nu : \mathbf{C} \times \mathbf{D} \to \mathbf{C}$ whose unit and Kleisli lifting we denote η^ν and $-^{\bigstar}$, respectively. Moreover,

1. *If $\#$ is guarded then so is $\#^\nu$ with guardedness defined as follows: given $\sigma : Y' \hookrightarrow Y$, $f : X \to Y \mathbin{\#^\nu} Z$ is σ-guarded if $\mathsf{out} f : X \to Y \mathbin{\#} (Y \mathbin{\#^\nu} Z, Z)$ is σ-guarded.*
2. *If $\#$ is pre-iterative under an iteration operator $-^\dagger$ then so is $\#^\nu$ with the iteration operator $-^\ddagger$ defined as follows:*

$$\big(f : X \to_2 (Y + X) \mathbin{\#^\nu} Z\big)^\ddagger = \mathsf{coit}\ \big([\eta, (\mathsf{out} f)^\dagger]^\star \mathsf{out}\big)\ \eta^\nu\ \mathsf{in}_2$$

3. *If $\#$ is iterative then so is $\#^\nu$ under the definition from the previous clause.*

Example 14. We spell out one instance of Theorem 13 in case $\mathbf{D} = 1$ and $\# = T(- + -)$ where $\mathbb{T} = (T, \eta, -^\star)$ is a monad. Then $\#^\nu$ is isomorphically a monad $\mathbb{T}^\nu$ on $\mathbf{C}$ with $T^\nu X = \nu\gamma.\, T(X + \gamma)$, unit $\eta^\nu = \eta\, \mathsf{in}_1$ and Kleisli star $(f : X \to T^\nu Y)^{\bigstar}$ being uniquely determined by the equation

$$\mathsf{out} f^{\bigstar} = [\mathsf{out} f,\ \eta\, \mathsf{in}_2 f^{\bigstar}]^\star \mathsf{out}.$$

If $\mathbb{T}$ is pre-iterative then so is $\mathbb{T}^\nu$ with iteration

$$(f : X \to T^\nu(Y + X))^\ddagger = \mathsf{coit}\ \big([\eta\, \mathsf{in}_2, (T[\mathsf{in}_1 + \mathsf{id},\ \mathsf{in}_1\, \mathsf{in}_2] \mathsf{out}\, f)^\dagger]^\star \mathsf{out}\big)\ \eta^\nu\ \mathsf{in}_2.$$

Example 15. Theorem 13 shows that our notion of guardedness extends along the applications of the final coalgebra transformer $\# \mapsto \#^\nu$ on parametrized monads. This can be used to capture existing notions of guardedness as follows. Consider $X \mathbin{\#} Y = T(X + (1 + A) \times Y)$ where $\mathbb{T}$ is some monad. In $\mathbf{Set}$, $1 + A$ can be thought of as consisting of a set A of visible actions and a silent action τ. In process algebra we standardly consider a process definition to be guarded if every recursive call is preceded by a visible action from A. In our framework this can be reconstructed as follows. The obvious isomorphism

$$\nu\gamma.\, T(X + (1 + A) \times \gamma) \cong \nu\gamma'.\, \nu\gamma.\, T(X + \gamma + A \times \gamma')$$

involves two more parametrized monads: $T(- + - + A \times -) : \mathbf{C} \times \mathbf{C} \times \mathbf{C} \to \mathbf{C}$ and $\nu\gamma.\, T(- + \gamma + A \times -) : \mathbf{C} \times \mathbf{C} \to \mathbf{C}$. By taking the latter to be trivially guarded and then defining guardedness for $\nu\gamma'.\, \nu\gamma.\, T(X + \gamma + A \times \gamma')$ using Theorem 13, we arrive precisely at the notion we expected for the isomorphic monad $\#^\nu$.

5 Complete Elgot Monads and Iteration Congruences

Besides the fixpoint identity we are interested in the following classical properties of the iteration operator, which we refer to as the *iteration laws* [7,10,27]:

- *naturality:* $g^\star f^\dagger = ([(T\,\mathsf{in}_1)\, g, \eta\,\mathsf{in}_2]^\star\, f)^\dagger$ for $f : X \to_2 T(Y + X)$, $g : Y \to TZ$;
- *dinaturality:* $([\eta\,\mathsf{in}_1, h]^\star\, g)^\dagger = [\eta, ([\eta\,\mathsf{in}_1, g]^\star\, h)^\dagger]^\star\, g$ for $g : X \to_2 T(Y + Z)$ and $h : Z \to T(Y + X)$ or $g : X \to T(Y + Z)$ and $h : Z \to_2 T(Y + X)$;
- *codiagonal:* $(T[\mathsf{id}, \mathsf{in}_2]\, f)^\dagger = f^{\dagger\dagger}$ for $f : X \to_{12,2} T((Y + X) + X)$;
- *uniformity:* $f\, h = T(\mathsf{id} + h)\, g$ implies $f^\dagger\, h = g^\dagger$ for $f : X \to_2 T(Y + X)$, $g : Z \to_2 T(Y + Z)$ and $h : Z \to X$.

The axioms are summarized in graphical form in Fig. 2, and then become quite intuitive. The two versions of the dinaturality axiom correspond to the alternative sets of guardedness assumptions mentioned above. We indicate the scope of the iteration operator by a shaded box and guardedness by bullets at the outputs of a morphism.

A guarded pre-iterative monad is called a *complete Elgot monad* if it is totally guarded and satisfies all iteration laws. In the sequel we shorten 'complete Elgot monads' to 'Elgot monads' (to be distinguished from Elgot monads in the sense of [2], which have solutions only for morphisms with finitely presentable domain).

In general, the fact that the iteration laws are correctly formulated relies on the axioms for guardedness. E.g., in the dinaturality axiom it suffices to assume that $g : X \to T(Y + Z)$ is in_2-guarded and this implies that both $[\eta\,\mathsf{in}_1, h]^\star\, g$ and $[\eta\,\mathsf{in}_1, g]^\star\, h$ are in_2-guarded by **(cmp)** and **(trv)**, and additionally **(sum)** in the latter case. Symmetrically, it suffices to make the analogous assumption about h. In the codiagonal axiom, it follows from the assumption $f : X \to_{12,2} T((Y + X) + X)$ by **(cmp)** that $T[\mathsf{id}, \mathsf{in}_2]\, f$ is in_2-guarded and by Proposition 6 that $f^\dagger$ is in_2-guarded. Indeed, the axioms for guarded monads are designed precisely to enable the formulation of the iteration laws.

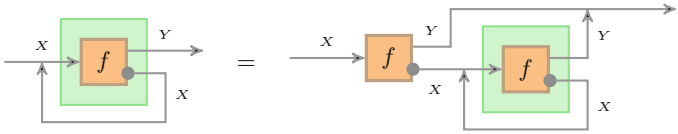
We show that for guarded iterative monads all iteration laws are automatic. Prior to that, we show that dinaturality follows from the others (thus generalizing the corresponding observation made recently [13,15]).

Proposition 16. *Any guarded pre-iterative monad satisfying naturality, codiagonal and uniformity also satisfies dinaturality, as well as the* Bekić identity

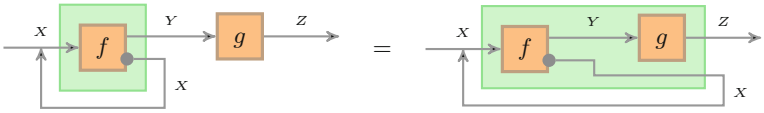
$$\left[T[\mathsf{id} + \mathsf{in}_1, \mathsf{in}_2]^\star\, f, T[\mathsf{id} + \mathsf{in}_1, \mathsf{in}_2]^\star\, g\right]^\dagger = [h^\dagger, [\eta, h^\dagger]^\star g^\dagger]$$

where $f : X \to_{12,2} T((Y + X) + Z)$, $g : Z \to_{12,2} T((Y + X) + Z)$, *and* $h = [\eta, g^\dagger]^\star f : X \to_2 T(Y + X)$.

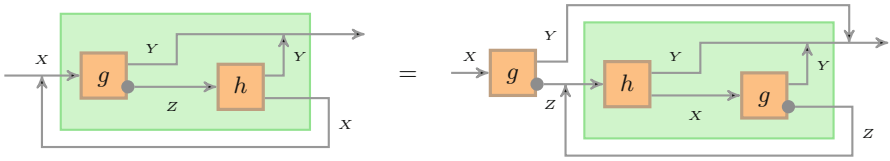
Fixpoint:



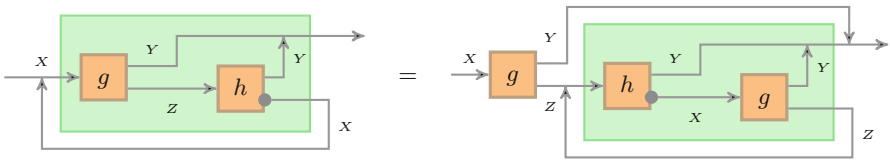
Naturality:



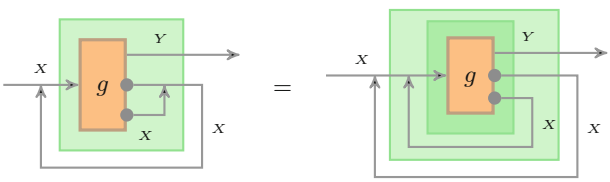
Dinaturality 1:



Dinaturality 2:



Codiagonal:



Uniformity:

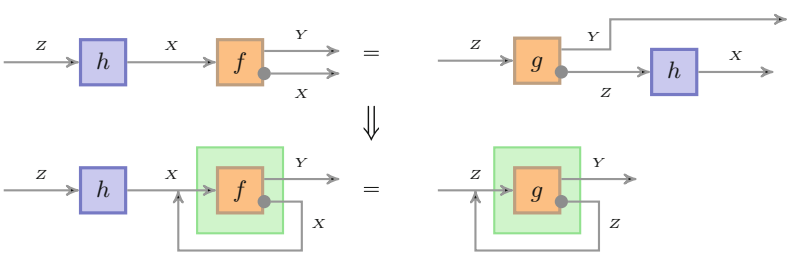


Fig. 2. Axioms of guarded iteration.

The proof of the following result runs in accordance with the original ideas of Elgot [10] for iterative theories, except that, by Proposition 16, dinaturality is now replaced by uniformity.

Theorem 17. *Every guarded iterative monad validates naturality, dinaturality, codiagonal and uniformity.*

We now proceed to introduce key properties of morphisms of guarded monads that allow for transferring pre-iterativity and the iteration laws, respectively.

Definition 18 (Guarded retraction). Let $\mathbb{T}$ and $\mathbb{S}$ be guarded monads. We call a monad morphism $\rho : \mathbb{T} \to \mathbb{S}$ a *guarded retraction* if there is a family of morphisms $(\upsilon_X : SX \to TX)_{X \in |\mathbf{C}|}$ (not necessarily natural in X!) such that

1. for every $f : X \to_\sigma SY$, we have $\upsilon_Y f : X \to_\sigma TY$,
2. $\rho_X \upsilon_X = \mathsf{id}$ for all $X \in |\mathbf{C}|$.

Theorem 19. *Let $\rho : \mathbb{T} \to \mathbb{S}$ be a guarded retraction, witnessed by $\upsilon : \mathbb{S} \to \mathbb{T}$, and suppose that $(\mathbb{T}, -^\dagger)$ is guarded pre-iterative. Then $\mathbb{S}$ is guarded pre-iterative with the iteration operator $(-)^\ddagger$ given by $f^\ddagger = \rho\,(\upsilon f)^\dagger$.*

Definition 20 (Iteration congruence). Let $\mathbb{T}$ be a guarded pre-iterative monad and let $\mathbb{S}$ be a monad. We call a monad morphism $\rho : \mathbb{T} \to \mathbb{S}$ an *iteration congruence* if for every pair of morphisms $f, g : X \to_2 T(Y + X)$,

$$\rho f = \rho g \implies \rho f^\dagger = \rho g^\dagger. \tag{2}$$

If ρ is moreover a guarded retraction, we call ρ an *iteration-congruent retraction.*

Theorem 21. *Under the premises of Theorem 19, assume moreover that ρ is an iteration-congruent retraction. Then any property out of naturality, dinaturality, codiagonal, and uniformity that is satisfied by $\mathbb{T}$ is also satisfied by $\mathbb{S}$.*

Proof (Sketch). The crucial observation is that under our assumptions, (2) is equivalent to the condition that for all $f : X \to_2 T(Y + X)$,

$$\rho\,(\upsilon\rho\, f)^\dagger = \rho f^\dagger. \tag{3}$$

Indeed, (2) $\implies$ (3), for $\rho\upsilon\rho\, f = \rho\, f$ and therefore $\rho(\upsilon\rho\, f)^\dagger = \rho\, f^\dagger$ and conversely, assuming (3) both for f and for g, and $\rho f = \rho g$, we obtain that $\rho f^\dagger = \rho(\upsilon\rho\, f)^\dagger = \rho(\upsilon\rho\, g)^\dagger = \rho g^\dagger$. Using (3), the claim is established routinely. □

Recall from the introduction that a monad $\mathbb{S}$ is *iteratable* if its coinductive resumption transform $\mathbb{S}^\nu$ exists. We make $\mathbb{S}^\nu$ into a guarded monad by applying Theorem 13 to $\mathbb{S}$ as a trivially guarded monad; explicitly: $f : X \to S^\nu(Y + X)$ is guarded iff $\mathsf{out}\, f = S(\mathsf{in}_1 + \mathsf{id})\, g$ for some $g : X \to S(Y + S^\nu(Y + X))$. We are now set to prove our first main result, which states that every iteratable Elgot monad can be obtained by quotienting a guarded iterative monad; that is, every choice of solutions that obeys the iteration laws arises by quotienting a more fine-grained model in which solutions are uniquely determined:

Theorem 22. *A totally guarded iteratable monad* $\mathbb{S}$ *is an Elgot monad iff there is a guarded iterative monad* $\mathbb{T}$ *and an iteration-congruent retraction* $\rho : \mathbb{T} \to \mathbb{S}$. *Specifically, every iteratable Elgot monad* $\mathbb{S}$ *is an iteration-congruent retract of its coinductive resumption transform* $\mathbb{S}^\nu$.

Proof (Sketch). Direction ($\Leftarrow$) immediately follows from Theorems 17 and 21.

In order to prove ($\Rightarrow$), we show that $\mathbb{S} = (S, \eta, -^\star, -^\dagger)$ is an iteration-congruent retract of $\mathbb{S}^\nu = (\nu\gamma.\, S(- + \gamma), \eta^\nu, -^\star, -^\ddagger)$. Let $\upsilon_X = \mathsf{out}^{-1}\, \eta\, \mathsf{in}_2\, \mathsf{out}^{-1}(S\, \mathsf{in}_1)$ and

$$\rho_X = \big(S^\nu X \xrightarrow{\mathsf{out}} S(X + TX)\big)^\dagger.$$

Clearly, υf is σ-guarded for every $f : X \to SY$ and it is easy to verify that υ is left inverse to ρ by using the fixpoint identity for $-^\dagger$ twice.

Naturality of ρ is proved straightforwardly from naturality of $-^\dagger$. The remaining calculations showing that ρ is a monad morphism and moreover an iteration congruence make heavy use of the Elgot monad laws. □

The notions of guarded retraction and iteration congruence straightforwardly extend to parametrized monads. We then can take the claims of Theorem 13 further.

Theorem 23. *Let* $\#, \hat{\#} : \mathbf{C} \times (\mathbf{C} \times \mathbf{D}) \to \mathbf{C}$ *be guarded parametrized monads and let* $\rho : \# \to \hat{\#}$ *be an iteration-congruent retraction. By Theorem 13,* $\#^\nu = - \#(\gamma, -)$ *and* $\hat{\#}^\nu = - \hat{\#}(\gamma, -)$ *are also parametrized guarded monads. Then* $\rho^\nu : \#^\nu \to \hat{\#}^\nu$, *with components*

$$\rho^\nu_{X,Y} = \mathsf{coit}\, \big(\nu\gamma.\, X \# (\gamma, Y) \xrightarrow{\rho\mathsf{out}} X \hat{\#} (\nu\gamma.\, X \# (\gamma, Y), Y)\big),$$

is again an iteration-congruent retraction.

Theorems 22 and 23 jointly provide a simple and structured way of showing that Elgotness extends along the parametrized monad transformer $\# \mapsto \hat{\#}$: If $- \#X$ is Elgot then by Theorem 22 there is an iteration-congruent retraction $\rho : \nu\gamma.\, - + \gamma \# X \to - \#X$; by Theorem 23, it gives rise to an iteration-congruent retraction

$$\rho^\nu : \nu\gamma'.\, \nu\gamma.\, - + \gamma \# (\gamma', X) \to \nu\gamma'.\, - \#(\gamma', X)$$

and by Theorem 22, the right-hand side is again Elgot. We have thus proved.

Corollary 24. *Given a parametrized monad* $\#$ *and* $X \in |\mathbf{C}|$, *if* $- \#X$ *is Elgot then so is* $- \#^\nu X = \nu\gamma.\, - \#(\gamma, X)$.

This yields a more general and simpler proof of one of the main results in [15].

Example 25. By instantiating $\#$ in Corollary 24 with $\mathcal{P}_\omega(- + A \times -) : \mathbf{Set} \times \mathbf{Set} \to \mathbf{Set}$ where $\mathcal{P}_\omega$ is the countable powerset monad, we obtain $\nu\gamma.\, \mathcal{P}_\omega(X + A \times \gamma)$, which can be viewed as a semantic domain for countably branching processes that possibly terminate with results in X and are taken modulo strong bisimilarity. The simple fact that $\mathcal{P}_\omega$ is Elgot [15] implies that so is $\nu\gamma.\, \mathcal{P}_\omega(X + A \times \gamma)$.

This justifies the use of systems of possibly unguarded recursive process algebra equations (as done, e.g., in [6]). It is worth noting that the iteration operator of the transformed monad is neither least nor unique. It arises by introducing an additional delay action that guards all recursive calls and then eliminating these delays from the unique solution of the new recursive definition; the delay elimination is the effect of ρ^ν in Theorem 23.

Theorem 22 characterizes iteratable Elgot monads as iteration-congruent retracts of their $(-)^\nu$-transforms. We take this perspective further as follows. Let us call $\mathbb{T}$ *strongly iteratable* if every $T^{\nu\cdots\nu}$ exists. Consider the functor $\mathbb{T} \mapsto \mathbb{T}^\nu$ on the category of strongly iteratable monads over $\mathbf{C}$. This is itself a monad: the unit $\boldsymbol{\eta}$ is the natural transformation with components $\boldsymbol{\eta}_X = \mathsf{out}^{-1}(T\,\mathsf{in}_1) : TX \to T^\nu X$ and the multiplication $\boldsymbol{\mu} : T^{\nu\nu} \to T^\nu$ has components

$$\boldsymbol{\mu}_X = \mathsf{coit}\,\big(T[\mathsf{id},\, \mathsf{in}_2\,\mathsf{out}^{-1}]\mathsf{out}\,\mathsf{out} : T^{\nu\nu}X \to T(X + T^{\nu\nu}X)\big).$$

For every T we define the *delay transformation* $\triangleright = \mathsf{out}^{-1}\,\eta\,\mathsf{in}_2 : T^\nu \to T^\nu$. This leads to our second main result:

Theorem 26. *The category of strongly iteratable Elgot monads over* $\mathbf{C}$ *is isomorphic to the full subcategory of the category of* $(-)^\nu$*-algebras for strongly iteratable* $\mathbb{S}$ *consisting of the* $(-)^\nu$*-algebras* $(S^\nu, \rho : S^\nu \to S)$ *satisfying* $\rho \triangleright = \rho$.

Proof (Sketch). To show that every strongly iteratable Elgot monad is a $(-)^\nu$-algebra, one has to check the equations $\rho\boldsymbol{\eta} = \mathsf{id}$ and $\rho\boldsymbol{\mu} = \rho\rho^\nu$ where $\rho^\nu = \mathsf{coit}\,(\rho\,\mathsf{out}) : S^{\nu\nu} \to S^\nu$. The first equation follows relatively easily. The second one is shown along the following lines:

$$\rho\boldsymbol{\mu} \overset{\text{(i)}}{=\!=} \rho[\eta,\, (\triangleright\,\mathsf{out})^\ddagger]^\star\mathsf{out} \overset{\text{(ii)}}{=\!=} \rho\big(\mathsf{out}^{-1}\,S(\,\mathsf{in}_1 + \eta^\nu\,\mathsf{in}_2)\rho\mathsf{out}\big)^\ddagger \overset{\text{(iii)}}{=\!=} \rho\rho^\nu.$$

Here, (i) and (iii) only amount to equivalent transformations of $\boldsymbol{\mu}$ and ρ^ν, respectively, while (ii) makes crucial use of the fact that ρ is an iteration congruence, as implied by Theorem 22.

For the converse implication, we start with a $(-)^\nu$-algebra and verify the Elgot monad laws for the iteration operator $f^\dagger = \rho(\mathsf{coit}\ f)$. □

Remark 27. The delay cancellation condition $\rho \triangleright = \rho$ is essential, as can be seen on a simple example. Let $\mathbf{Mon}(\mathbf{C})^\nu$ be the category of $(-)^\nu$-algebras and let $\mathbf{Mon}(\mathbf{C})^\nu_\triangleright$ be its subcategory figuring in Theorem 26. Since the identity functor is the initial monad, the initial object of $\mathbf{Mon}(\mathbf{C})^\nu$ is Capretta's delay monad [8] $D = \nu\gamma.(-+\gamma)$. On the other hand, the initial object of $\mathbf{Mon}(\mathbf{C})^\nu_\triangleright$ (if it exists) is the *initial Elgot monad* $\mathbb{L}$, which on $\mathbf{C} = \mathbf{Set}$ is the *maybe monad* $(-) + 1$.

If $\mathbf{C} = \mathbf{Set}$, then $DX = (X \times N + 1)$ does turn out to be Elgot [14] (but applying Theorem 26 to D qua Elgot monad yields a different $(-)^\nu$-algebra structure than the initial one), and $\mathbb{L}$ is, in this case, a retract of $\mathbb{D}$ in $\mathbf{Mon}(\mathbf{C})^\nu_\triangleright$. The situation is more intricate in categories with a non-classical internal logic, for which $\mathbb{D}$ is mainly intended. We believe that in such a setting, neither is $\mathbb{D}$ Elgot in general, nor is $\mathbb{L}$ the maybe monad. However, there will still be a unique $(-)^\nu$-algebra morphism $\mathbb{D} \to \mathbb{L}$ in $\mathbf{Mon}(\mathbf{C})^\nu$.

6 A Sandwich Theorem for Elgot Monads

As an application of Theorem 22, we show that sandwiching an Elgot monad between adjoint functors again yields an Elgot monad. A similar result has been shown for completely iterative monads [26]; this result generalizes straightforwardly to guarded iterative monads:

Theorem 28. *Let $F : \mathbf{C} \to \mathbf{D}$ and $U : \mathbf{D} \to \mathbf{C}$ be a pair of adjoint functors with the associated natural isomorphism $\Phi : \mathbf{D}(FX, Y) \to \mathbf{C}(X, UY)$, and let $\mathbb{T}$ be a guarded iterative monad on $\mathbf{D}$. Then the monad induced on the composite functor UTF is guarded iterative, with the guardedness relation defined by taking $f : X \to_\sigma UTFY$ if and only if $\Phi^{-1} f : FX \to_\sigma TFY$, and the unique solutions given by $f \mapsto \Phi((\Phi^{-1} f)^\dagger)$.*

Now, to obtain a similar result for Elgot monads, we can easily combine Theorems 22 and 28 without having to verify the equational properties by hand.

Theorem 29. *With an adjunction as in Theorem 28, let $\mathbb{S}$ be an Elgot monad on $\mathbf{D}$. Then, the monad induced on the composition USF is an Elgot monad.*

Proof (Sketch). By Theorem 22, there exists a guarded iterative monad $\mathbb{T}$ and an iteration-congruent retraction $\rho : \mathbb{T} \to \mathbb{S}$. By Theorem 28, the monad induced on UTF is guarded iterative. Again by Theorem 22, it suffices to show that $U\rho F : UTF \to USF$ is an iteration-congruent retraction, which is straightforward. □

Example 30 (From Metric to CPO-based Iteration). As an example exhibiting sandwiching as well as the setting of Theorem 22, we compare two iteration operators on **Set** that arise from different fixed-point theorems: Banach's, for complete metric spaces, and Kleene's, for complete partial orders, respectively. We obtain the first operator by sandwiching Escardo's *metric lifting monad* $\mathbb{S}$ [12] in the adjunction between sets and bounded complete ultrametric spaces (which forgets the metric in one direction and takes discrete spaces in the other), obtaining a monad $\bar{\mathbb{S}}$ on **Set**. Given a bounded complete metric space (X, d), $S(X, d)$ is a metric on the set $(X \times \mathbb{N}) \cup \{\bot\}$. As we show in the appendix, $\mathbb{S}$ is guarded iterative if we define $f : (X, d) \to S(Y, d')$ to be σ-guarded if $k > 0$ whenever $f(x) = (\sigma(y), k)$. By Theorem 28, $\bar{\mathbb{S}}$ is also guarded iterative (of course, this can also be shown directly). The second monad arises by sandwiching the identity monad on cpos with bottom in the adjunction between sets and cpos with bottom that forgets the ordering in one direction and adjoins bottom in the other, obtaining an Elgot monad $\mathbb{L}$ on Set according to Theorem 29. The latter is unsurprising, of course, as $\mathbb{L}$ is just the maybe monad $LX = X + 1$.

The monad $\bar{\mathbb{S}}$ keeps track of the number of steps needed to obtain the final result. We have an evident extensional collapse map $\rho : \bar{\mathbb{S}} \to \mathbb{L}$, which just forgets the number of steps. One can show that ρ is in fact an iteration-congruent retraction, so we obtain precisely the situation of Theorem 22.

7 Related Work

Alternatively to our guardedness relation on Kleisli morphisms, guardedness can be formalized using type constructors [23] or, categorically, functors, as in *guarded fixpoint categories* [20]; the latter cover also total guardedness, like we do. Our approach is slightly more fine-grained, and in particular natively supports the two variants of the dinaturality axiom (Fig. 2), which, e.g., in guarded fixpoint categories require additional assumptions [20, Proposition 3.15] akin to the one we discuss in Remark 4.

A result that resembles our Theorem 26, due to Adámek et al. [3], states roughly that if $\mathbf{C}$ is locally finitely presentable and hyperextensive (e.g. $\mathbf{C} = \mathbf{Set}$) then the finitary Elgot monads are the algebras for a monad on the category of endofunctors given by $H \mapsto L_H = \rho\gamma.(- + 1 + H\gamma)$ where ρ takes *rational fixpoints* (i.e. final coalgebras among those where every point generates a finite subcoalgebra). Besides Theorem 26 making fewer assumptions on $\mathbf{C}$, the key difference is that, precisely by dint of this result, L_H is already an Elgot monad; contrastingly, we characterize Elgot monads as quotients of *guarded iterative* monads, i.e. of monads where guarded recursive definitions have *unique* fixpoints.

8 Conclusions and Further Work

We have given a unified account of monad-based guarded and unguarded iteration by axiomatizing the notion of guardedness to cover standard definitions of guardedness, and additionally, as a corner case, what we call *total guardedness*, i.e. the situation when all morphisms are declared to be guarded. We thus obtain a common umbrella for *guarded iterative monads*, i.e. monads with unique iterates of guarded morphisms, and Elgot monads, i.e. totally guarded monads satisfying Elgot's classical laws of iteration. We reinforce the view that the latter constitute a canonical model for monad-based unguarded iteration by establishing the following equivalent characterizations: provided requisite final coalgebras exist, a monad $\mathbb{T}$ is Elgot iff

- it satisfies the quasi-equational theory of iteration [2,15] (definition);
- it is an iteration-congruent retract of a guarded iterative monad (Theorem 22);
- it is an algebra $(\mathbb{T}, \rho)$ of the monad $T \mapsto \nu\gamma.T(X + \gamma)$ in the category of monads satisfying a natural delay cancellation condition (Theorem 26).

In future work, we aim to investigate further applications of this machinery, in particular to examples which did not fit previous formalizations. One prospective target is suggested by the work of Nakata and Uustalu [24], who give a coinductive big-step trace semantics for a while-language. We conjecture that this work has an implicit guarded iterative monad $\mathbb{T}_{\mathbb{R}}$ under the hood, for which guardedness cannot be defined using the standard argument based on a final coalgebra structure of the monad because $\mathbb{T}_{\mathbb{R}}$ is not a final coalgebra.

In type theory, there is growing interest in forming an extensional quotient of the delay monad [4,9]. It is shown in [9] that under certain reasonable conditions, a suitable collapse of the delay monad by removing delays is again a

monad; however, the proof is already quite complex, and proving directly that the collapse is in fact an Elgot monad, as one would be inclined to expect, seems daunting. We expect that Theorem 26 may shed light on this issue. A natural question that arises in this regard is whether the subcategory of $(-)^\nu$-algebras figuring in the theorem is reflexive. A positive answer would provide a means of constructing canonical quotients of $(-)^\nu$-algebras (such as the delay monad) with the results automatically being Elgot monads.

References

1. Aczel, P., Adámek, J., Milius, S., Velebil, J.: Infinite trees and completely iterative theories: a coalgebraic view. Theoret. Comput. Sci. **300**(1–3), 1–45 (2003)
2. Adámek, J., Milius, S., Velebil, J.: Equational properties of iterative monads. Inf. Comput. **208**, 1306–1348 (2010)
3. Adámek, J., Milius, S., Velebil, J.: Elgot theories: a new perspective of the equational properties of iteration. Math. Struct. Comput. Sci. **21**, 417–480 (2011)
4. Altenkirch, T., Danielsson, N.: Partiality, revisited. In: Types for Proofs and Programs, TYPES 2016 (2016)
5. Atkey, R.: Parameterised notions of computation. J. Funct. Prog. **19**, 335 (2009)
6. Bergstra, J., Ponse, A., Smolka, S. (eds.): Handbook of Process Algebra. Elsevier, Amsterdam (2001)
7. Bloom, S., Ésik, Z.: Iteration Theories: The Equational Logic of Iterative Processes. Springer, Heidelberg (1993)
8. Capretta, V.: General recursion via coinductive types. Log. Meth. Comput. Sci. **1**(2), 1–28 (2005)
9. Chapman, J., Uustalu, T., Veltri, N.: Quotienting the delay monad by weak bisimilarity. In: Leucker, M., Rueda, C., Valencia, F.D. (eds.) ICTAC 2015. LNCS, vol. 9399, pp. 110–125. Springer, Cham (2015). doi:10.1007/978-3-319-25150-9_8
10. Elgot, C.: Monadic computation and iterative algebraic theories. In: Rose, H.E., Shepherdson, J.C. (eds.) Logic Colloquium 1973. Studies in Logic and the Foundations of Mathematics, vol. 80, pp. 175–230. Elsevier, Amsterdam (1975)
11. Elgot, C., Bloom, S., Tindell, R.: On the algebraic atructure of rooted trees. J. Comput. Syst. Sci. **16**, 362–399 (1978)
12. Escardó, M.H.: A metric model of PCF. In: Realizability Semantics and Applications (1999)
13. Ésik, Z., Goncharov, S.: Some remarks on Conway and iteration theories. CoRR, abs/1603.00838 (2016)
14. Goncharov, S., Milius, S., Rauch, C.: Complete Elgot monads and coalgebraic resumptions. In: Mathematical Foundations of Programming Semantics, MFPS 2016. ENTCS (2016)
15. Goncharov, S., Rauch, C., Schröder, L.: Unguarded recursion on coinductive resumptions. In: Mathematical Foundations of Programming Semantics, MFPS 2015. ENTCS (2015)
16. Goncharov, S., Schröder, L.: A coinductive calculus for asynchronous side-effecting processes. Inf. Comput. **231**, 204–232 (2013)
17. Lawvere, W.: Functorial semantics of algebraic theories. Proc. Natl. Acad. Sci. USA **50**, 869–872 (1963)
18. Mac Lane, S.: Categories for the Working Mathematician. Springer, Heidelberg (1971)

19. Milius, S.: Completely iterative algebras and completely iterative monads. Inf. Comput. **196**, 1–41 (2005)
20. Milius, S., Litak, T.: Guard your daggers and traces: properties of guarded (co-)recursion. Fund. Inform. **150**, 407–449 (2017)
21. Moggi, E.: A modular approach to denotational semantics. In: Pitt, D.H., Curien, P.-L., Abramsky, S., Pitts, A.M., Poigné, A., Rydeheard, D.E. (eds.) CTCS 1991. LNCS, vol. 530, pp. 138–139. Springer, Heidelberg (1991). doi:10.1007/BFb0013462
22. Moggi, E.: Notions of computation and monads. Inf. Comput. **93**, 55–92 (1991)
23. Nakano, H.: A modality for recursion. In: Logic in Computer Science, LICS 2000, pp. 255–266. IEEE Computer Society (2000)
24. Nakata, K., Uustalu, T.: A Hoare logic for the coinductive trace-based big-step semantics of while. Log. Meth. Comput. Sci. **11**(1), 1–32 (2015)
25. Piróg, M., Gibbons, J.: The coinductive resumption monad. In: Mathematical Foundations of Programming Semantics, MFPS 2014. ENTCS, vol. 308, pp. 273–288 (2014)
26. Piróg, M., Gibbons, J.: Monads for behaviour. In: Mathematical Foundations of Programming Semantics, MFPS 2013. ENTCS, vol. 298, pp. 309–324 (2015)
27. Simpson, A., Plotkin, G.: Complete axioms for categorical fixed-point operators. In: Logic in Computer Science, LICS 2000, pp. 30–41 (2000)
28. Uustalu, T.: Generalizing substitution. ITA **37**, 315–336 (2003)
29. Uustalu, T., Vene, V.: Primitive (co)recursion and course-of-value (co)iteration, categorically. Informatica **10**(1), 5–26 (1999). Lithuanian Academy of Sciences

Partiality, Revisited

The Partiality Monad as a Quotient Inductive-Inductive Type

Thorsten Altenkirch[1(✉)], Nils Anders Danielsson[2(✉)], and Nicolai Kraus[1(✉)]

[1] University of Nottingham, Nottingham, UK
{thorsten.altenkirch,nicolai.kraus}@nottingham.ac.uk
[2] University of Gothenburg, Gothenburg, Sweden
nad@cse.gu.se

Abstract. Capretta's delay monad can be used to model partial computations, but it has the "wrong" notion of built-in equality, strong bisimilarity. An alternative is to quotient the delay monad by the "right" notion of equality, weak bisimilarity. However, recent work by Chapman et al. suggests that it is impossible to define a monad structure on the resulting construction in common forms of type theory without assuming (instances of) the axiom of countable choice.

Using an idea from homotopy type theory—a higher inductive-inductive type—we construct a partiality monad without relying on countable choice. We prove that, in the presence of countable choice, our partiality monad is equivalent to the delay monad quotiented by weak bisimilarity. Furthermore we outline several applications.

1 Introduction

Computational effects can be modelled using monads, and in some functional programming languages (notably Haskell) they are commonly used as a program structuring device. In the presence of dependent types one can both write and reason about monadic programs. From a type theorist's point of view, even a "pure" functional language like Haskell is not really pure as it has built-in effects, one of which is partiality: a function does not necessarily terminate. It is thus natural to look for a partiality monad which makes it possible to model partial computations and to reason about possibly non-terminating programs.

Capretta modeled partial computations using a coinductive construction that we call the *delay monad* [6]. We use the notation $\mathrm{D}(A)$ for Capretta's type of delayed computations over a type A. $\mathrm{D}(A)$ is coinductively generated by $\mathsf{now} : A \to \mathrm{D}(A)$ and $\mathsf{later} : \mathrm{D}(A) \to \mathrm{D}(A)$. Examples of elements of $\mathrm{D}(A)$ include $\mathsf{now}(a)$ and $\mathsf{later}(\mathsf{later}(\mathsf{now}(a)))$, as well as the infinitely delayed value $\bot$, defined

T. Altenkirch—Supported by EPSRC grant EP/M016994/1 and by USAF, Airforce office for scientific research, award FA9550-16-1-0029.

N.A. Danielsson—Supported by a grant from the Swedish Research Council (621-2013-4879).

N. Kraus—Supported by EPSRC grant EP/M016994/1.

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 534–549, 2017.
DOI: 10.1007/978-3-662-54458-7_31

by the guarded equation $\bot = \mathsf{later}(\bot)$. We can model recursive programs as Kleisli arrows $A \to \mathrm{D}(B)$, and we can construct fixpoints of (ω-continuous) functions of type $(A \to \mathrm{D}(B)) \to (A \to \mathrm{D}(B))$, see Benton et al. [5].

Unfortunately, Capretta's delay monad is sometimes too intensional. It is often appropriate to treat two computations as equal if they terminate with the same value, but the delay monad allows us to count the number of "steps" (later constructors) used by a computation.

Capretta addressed this problem by defining a relation that we call *weak bisimilarity*, $\sim_{\mathrm{D}}$, and that relates expressions that only differ by a finite number of later constructors [6]. Capretta proved that the delay monad combined with weak bisimilarity is a monad in the category of setoids.

A setoid is a pair consisting of a type and an equivalence relation on that type. Setoids are sometimes used to approximate quotient types in type theories that lack support for quotients. However, a major difference between setoids and quotient types is that setoids do not provide a mechanism for information hiding. Using the setoid approach basically boils down to introducing a new relation together with the convention that all constructions have to respect this relation. A problem with this approach is that it can lead to something which has informally become known as *setoid hell*, in which one is forced to prove that a number of constructions—even some that do not depend on implementation details by, say, pattern matching on the now and later constructors—preserve setoid relations. This kind of problem does not afflict quotient types.

In a type theory with quotient types [14], one can consider using the quotient $\mathrm{D}(A)/{\sim_{\mathrm{D}}}$ as the type of partial computations of type A. This idea was discussed in a talk by Uustalu [7], reporting on joint work with Capretta and the first-named author of the current paper. However, the idea does not seem to work as intended. It is an open problem—and believed to be impossible—to show that this construction actually constitutes a monad (in "usual" forms of type theory).

With an additional assumption, Chapman et al. have managed to show that the partiality operator $\mathrm{D}(-)/{\sim_{\mathrm{D}}}$ is a monad [8]. This additional assumption is known as *countable choice*. To express what this is, first note that the *propositional truncation*, written $\|-\|$ and sometimes called "squashing", is an operation that turns a type into a proposition (a type with at most one element). We can see $\|A\|$ as the quotient of A by the total relation. Countable choice says that Π and $\|-\|$ commute if the domain is the natural numbers, in the sense that there is a function from $\Pi_{n:\mathbb{N}} \|P(n)\|$ to $\|\Pi_{n:\mathbb{N}} P(n)\|$. Even though this principle holds in some models, its status in type theory is unclear: the principle is believed to be independent of several variants of type theory. Recently Coquand et al. have shown that it cannot be derived in a theory with propositional truncation and a single univalent universe [10], speculating that the result might extend to a theory with a hierarchy of universes. Furthermore Richman argues that countable choice should be avoided in constructive reasoning [18]. The main purpose of the present paper is to define a partiality monad without making use of this principle.

The situation with the quotiented delay monad is similar to that of one variant of the real numbers in constructive mathematics. If the Cauchy reals are

defined as a quotient, then it is impossible to prove a specific form of the statement that every Cauchy sequence of Cauchy reals has a limit using $\mathsf{IZF}_{\mathrm{Ref}}$, a constructive set theory without countable choice [17]. It is suspected that corresponding statements are also impossible to prove in several variants of type theory. An alternative solution was put forward in the context of homotopy type theory [20]. In that approach, the reals are constructed inductively simultaneously with a notion of closeness, and the quotienting is done directly in the definition using a *higher inductive-inductive type* (HIIT).

In 2015, Andrej Bauer and the first-named author of the current paper suggested to use a similar approach to define a partiality monad without using countable choice, an idea which was mentioned by Chapman et al. [8]. Here, we show that this is indeed possible.

We do not make use of the full power of HIITs, but restrict ourselves to *set-truncated* HIITs. We call such types *quotient inductive-inductive types*, *QIITs*, following Altenkirch and Kaposi [1]. Some of the theory of QIITs is developed in the forthcoming PhD thesis of Dijkstra [12], see also Altenkirch et al. [2]. Although type theory extended with QIITs is still experimental and currently lacks a solid foundation, QIITs are a significantly simpler concept than full-blown HIITs. It is conjectured that QIITs exist in some computational models of type theory.

The type theory that we work in can be described as a fragment of the theory considered in the standard textbook on homotopy type theory [20] (henceforth referred to as the HoTT book), and is quite close to the theory considered by Chapman et al. [8]. Details are given in Sect. 2. The construction of our partiality monad is given in Sect. 3, together with its elimination principle and some properties. Furthermore we show that it gives us *free ω-cpos* in a sense that we will make precise. In Sect. 4 we show that, assuming countable choice, our partiality monad is equivalent to (in bijective correspondence to) the one of Chapman et al. [8]. We outline some applications of the partiality monad in Sect. 5, and conclude with a short discussion in Sect. 6.

Agda Formalisation. The paper is accompanied by a formal development [3] in Agda.

At the time of writing, Agda does not directly support QIITs. We have chosen to represent them by postulating their elimination principles together with the equalities they are supposed to satisfy. In some cases (but not for the partiality monad) we have also made use of Agda's experimental rewriting feature [9] to turn postulated equalities into judgmental computation rules.

Note that there are differences between the Agda code and the presentation in the text. For one, the formalisation discusses various additional topics that have been omitted in the paper for reasons of space, and is more rigorous. Furthermore, the paper defines the partiality monad's elimination principle as a universal property. In the formalisation the elimination principle is given as an induction principle, but we also prove that this principle is interderivable with the universal property. Finally there are a number of small differences between

the formalisation and the text, and some results in the paper have not been formalised at all, most notably the results about the reals in Sect. 5.2.

2 Background: Type Theory with Quotient Inductive-Inductive Types

We work in intensional type theory of Martin-Löf style with all the usual components (e.g. Π, Σ, inductive types), including the identity type (we use the notation $x = y$). We assume that equality of functions is extensional, and that (strong) bisimilarity implies equality for coinductive types.

Chapman et al. [8] assume the axiom of *uniqueness of identity proofs*, UIP, for all small types (types in the lowest universe). UIP holds for a type A if, for any elements x, $y : A$, if we have equalities p, $q : x = y$, then we have $p = q$. Instead of postulating an axiom, we prefer to work in a more general setting and restrict ourselves to types with the corresponding property. This approach is compatible with homotopy type theory. In the language of homotopy type theory, we work with *sets* or 0-*truncated types*; a type is a set if and only if it satisfies UIP. When we write $A : \mathsf{Set}$, we mean that A is a type (in some universe) with the property of being a set; and when we write $B : A \to \mathsf{Set}$, we mean that B is a family of types such that each $B(a)$ is a set.

Similarly to $A : \mathsf{Set}$, we write $P : \mathsf{Prop}$ for a type P with the property that it is a proposition, i.e. a (-1)-truncated type, i.e. a type with the property that any two of its elements are equal. A proposition is also a set. The type of all propositions in a certain universe is closed under all operations that are relevant to us, and the same applies to sets.

In addition to UIP, Chapman et al. [8] assume *propositional extensionality*—that logically equivalent propositions are equal—for all small propositions. This property is equivalent to the *univalence axiom* [20], restricted to (small) propositions. Just like Chapman et al., we only require propositional extensionality (not full univalence) for our development, with the exception that univalence is used to show that certain precategories (in the sense of the HoTT book [20]) are categories. For an example of how propositional extensionality is used, see Lemma 7.

Chapman et al. [8] also assume the existence of quotient types in the style of Hofmann [14]. Given a set A and a propositional relation $\sim$ on it, the (set-) quotient $A/\!\sim$ can in homotopy type theory be constructed as a higher inductive type with three constructors [20]:

$$\begin{aligned} [-] &: A \to A/\!\sim \\ [-]^{=} &: \Pi_{a,b:A}\, a \sim b \to [a] = [b] \\ \mathsf{irr} &: \Pi_{x,y:A/\sim}\, \Pi_{p,q:x=y}\, p = q \end{aligned}$$

The last constructor irr ensures that any two parallel equalities are equal, that is, that $A/\!\sim$ is set-truncated. We call a higher inductive type with such a set-truncation constructor a *quotient inductive type* (QIT).

As noted above, Chapman et al. [8] use countable choice, which we want to avoid. Instead we make use of quotient *inductive-inductive* types (QIITs) [2,12]. From the point of view of homotopy type theory, these are set-truncated higher inductive-inductive types (HIITs); some other examples of HIITs can be found in the HoTT book [20, Chap. 11]. While it seems plausible that QIITs exist in some computational models of type theory, this has yet to be determined.

3 The Partiality Monad

As indicated in the introduction we define the partiality monad $(-)_\perp$ as a QIIT, defining the type $A_\perp$ simultaneously with an ordering relation $\sqsubseteq$ on $A_\perp$. We will first describe the constructors and the elimination principle of this definition, and then show that $A_\perp$ is the underlying type of the free ω-cpo (see Definition 4) on A, thus proving that $(-)_\perp$ is a monad. In the final part of this section we will give a characterisation of the ordering relation; this is perhaps not as trivial as one might expect, given the relation's definition.

Note that our construction of $A_\perp$ can be seen as a further example of a free algebraic structure defined in type theory. It was discussed in the HoTT book [20, Chap. 6.11] that free groups can be defined as (in our terminology) quotient inductive types, while it is well-known that even simpler examples can be defined as ordinary inductive types.

3.1 The Definition and Its Elimination Principles

Let A be a set. We define the set $A_\perp$ simultaneously with a binary propositional relation on $A_\perp$, written $\sqsubseteq$. The set $A_\perp$ is generated by the following four constructors, plus a set-truncation constructor:

$$\begin{aligned} \eta &: A \to A_\perp & \bigsqcup &: (\Sigma_{s:\mathbb{N}\to A_\perp} \Pi_{n:\mathbb{N}}\, s_n \sqsubseteq s_{n+1}) \to A_\perp \\ \perp &: A_\perp & \alpha &: \Pi_{x,y:A_\perp}\, x \sqsubseteq y \to y \sqsubseteq x \to x = y \end{aligned}$$

The constructor η tells us that any element of A can be viewed as an element of $A_\perp$, and $\perp$ represents a non-terminating computation. The constructor $\bigsqcup$ is intended to form least upper bounds of increasing sequences, and α ensures that the ordering relation $\sqsubseteq$ is antisymmetric.

The relation $\sqsubseteq$ is a type family that is indexed twice by $A_\perp$. It is generated by six constructors. One of these constructors says that, for any $x, y : A_\perp$, the type $x \sqsubseteq y$ is a proposition ($\Pi_{p,q:x\sqsubseteq y}\, p = q$). Because any two proofs of $x \sqsubseteq y$ are equal, we do not name the constructors of the ordering relation. The remaining constructors are given as inference rules, where each rule is implicitly Π-quantified over its unbound variables (the same comment applies to other definitions below):

$$\frac{}{x \sqsubseteq x} \qquad \frac{x \sqsubseteq y \qquad y \sqsubseteq z}{x \sqsubseteq z} \qquad \frac{}{\perp \sqsubseteq x} \qquad \frac{}{\Pi_{n:\mathbb{N}}\, s_n \sqsubseteq \bigsqcup(s,p)} \qquad \frac{\Pi_{n:\mathbb{N}}\, s_n \sqsubseteq x}{\bigsqcup(s,p) \sqsubseteq x}$$

The rules state that $\sqsubseteq$ is reflexive and transitive, that $\bot$ is at least as small as any other element of $A_\bot$, and that $\bigsqcup$ constructs least upper bounds.

Now we will give the elimination principle of $(-)_\bot$ and $\sqsubseteq$. This principle can be stated in different ways. One way would be to state it as an induction principle, along the following lines: *Given a family* $P : A_\bot \to \mathsf{Set}$ *and [... something for* $\sqsubseteq$*...], and given elements of* $P(\bot)$, $\Pi_{a:A} P(\eta(a))$, *[... and so on...], we can conclude that* $\Pi_{x:A_\bot} P(x)$ *and [...].* We take this approach in our formalisation; for another example, see the presentation of the Cauchy reals in the HoTT book [20, Chap. 11.3.2]. However, because the two types are defined simultaneously and involve constructors targeting the equality type, the induction principle may look somewhat involved and perhaps even ad-hoc, and it may not be obvious that it is the "correct" one.

Instead, we present a universal property. Dijkstra [12] and Altenkirch et al. [2] have worked out a general form and rules for a large class of quotient inductive-inductive types. In their setting, any specification of a QIIT gives rise to a *category of algebras*, following methods that have been used for W-types [4] and certain higher inductive types [19], and if this category has a (homotopy-) initial object, then this object is taken as the definition of the QIIT. We use the following algebras:

Definition 1 (partiality algebras). *A* partiality algebra *over the set* A *consists of a set* X*; a propositional binary relation on* X, $\sqsubseteq_X$*; an element* $\bot_X : X$*, a family* $\eta_X : A \to X$*, and a family* $\bigsqcup_X : (\Sigma_{s:\mathbb{N}\to X} \Pi_{n:\mathbb{N}}\, s_n \sqsubseteq_X s_{n+1}) \to X$*; and the following laws:*

$$x \sqsubseteq_X x \qquad x \sqsubseteq_X y \to y \sqsubseteq_X z \to x \sqsubseteq_X z$$

$$\bot_X \sqsubseteq_X x \qquad x \sqsubseteq_X y \to y \sqsubseteq_X x \to x = y$$

$$\Pi_{n:\mathbb{N}}\, s_n \sqsubseteq_X \bigsqcup_X (s,p) \qquad (\Pi_{n:\mathbb{N}}\, s_n \sqsubseteq_X x) \to \bigsqcup_X (s,p) \sqsubseteq_X x$$

The type X *and the type family* $\sqsubseteq_X$ *are allowed to target universes distinct from the one that* A *lives in.*

For two partiality algebras over the same set A, $(X, \sqsubseteq_X, \bot_X, \eta_X, \bigsqcup_X)$ *and* $(Z, \sqsubseteq_Z, \bot_Z, \eta_Z, \bigsqcup_Z)$*, a* morphism of partiality algebras *from the former to the latter consists of a function* $f : X \to Z$ *satisfying the following laws: First,* f *has to respect the ordering relation,* $f^\sqsubseteq : x \sqsubseteq_X y \to f(x) \sqsubseteq_Z f(y)$*. Second,* f *has to preserve some of the constructors,* $f(\bot_X) = \bot_Z$, $f \circ \eta_X = \eta_Z$, *and* $f(\bigsqcup_X (s,p)) = \bigsqcup_Z (f \circ s, f^\sqsubseteq \circ p)$.

Let us denote this structure of objects and morphisms by Part_A.

The structure Part_A is a category, in which the identity morphism is the identity function, and composition of morphisms is composition of functions.

We can now make the elimination principle precise. Note that the tuple $(A_\bot, \sqsubseteq, \bot, \eta, \bigsqcup)$ is a partiality algebra. As the *elimination principle of* $A_\bot$ *and* $\sqsubseteq$ we take the statement that there is a unique (up to equality) morphism from this partiality algebra to any other partiality algebra over A. In the terminology of the HoTT book [20], the statement that there is a morphism is basically the

recursion principle of $(-)_\perp$ and $\sqsubseteq$, while uniqueness gives us the power of the induction principle (with *propositional* computation rules). Note that allowing the type X and the type family $\sqsubseteq_X$ to target arbitrary universes enables us to make use of large elimination.

We do not lose anything by using a universal property instead of an induction principle, at least for the induction principle referred to in the following theorem. The theorem is similar to results due to Dijkstra [12] and Altenkirch et al. [2]. It is stated without proof here, but a full proof of the fact can be found in our Agda development.

Theorem 2. *The elimination principle of $A_\perp$ and $\sqsubseteq$ can be stated as an induction principle. This induction principle, which comes with propositional rather than definitional computation rules, is interderivable with the universal property given above.*

Note that we could have defined $A_\perp$ and $\sqsubseteq$ differently. For instance, we could have omitted the set-truncation constructor from the definition of $A_\perp$, and then proved that the type is a set, following the approach taken for the Cauchy reals in the HoTT book [20]. However, if we had done this, then our definitions would have been less close to the general framework mentioned above [2,12].

As a simple demonstration of the universal property we construct an induction principle for $A_\perp$ that can be used when eliminating into a proposition. Following the terminology of the HoTT book [20, Chap. 11.3.2], we call it *partiality induction*:

Lemma 3. *Let P be a family of propositions on $A_\perp$ such that both $P(\perp)$ and $\Pi_{a:A}\, P(\eta(a))$ hold. Assume further that, for any increasing sequence $s : \mathbb{N} \to A_\perp$ (with corresponding proof p), $\Pi_{n:\mathbb{N}}\, P(s_n)$ implies $P(\bigsqcup(s, p))$. Then we can conclude $\Pi_{x:A_\perp}\, P(x)$.*

Proof. The proof uses a standard method. We define a partiality algebra where the set is $Z :\equiv \Sigma_{x:A_\perp}\, P(x)$; the binary relation is $\sqsubseteq$, ignoring the second projections of the values in Z; and the rest of the algebra is constructed using the assumptions. The universal property gives us a morphism m from the initial partiality algebra to this one, and in particular a function of type $A_\perp \to Z$. We are done if we can show that this function, composed with the first projection, is the identity on $A_\perp$. Note that the first projection can be turned into a partiality algebra morphism fst. Thus, by uniqueness, the composition of fst and m has to be the unique morphism from the initial partiality algebra to itself, and the function component of this morphism is the identity. □

3.2 ω-Complete Partial Orders

Another way of characterising our quotient inductive-inductive partiality monad is to say that $A_\perp$ is the *free (pointed) ω-cpo* over A:

Definition 4. *Let us denote the category $\mathsf{Part}_{\mathbf{0}}$, where $\mathbf{0}$ is the empty type, by ω-CPO. An ω-cpo is an object of this category.*

Let us quickly check that this definition makes sense. A partiality algebra on $\mathbf{0}$ is a set X with a binary propositional relation $\sqsubseteq_X$ that is a partial order. There is a least element $\bot_X$ and any increasing sequence has a least upper bound. There is also a function of type $\mathbf{0} \to X$, which we omit below as it carries no information.

We can now relate the category of sets [20, Example 9.1.7], written SET, to the category ω-CPO. For any ω-cpo we can take the underlying set, and it is easy to see that this yields a functor, in the sense of the HoTT book [20, Definition 9.2.1], $\mathsf{U} : \omega\text{-}\mathsf{CPO} \to \mathsf{SET}$.

We also have a functor $\mathsf{F} : \mathsf{SET} \to \omega\text{-}\mathsf{CPO}$, constructed as follows: The functor maps a set A to the ω-cpo $(A_\bot, \sqsubseteq, \bot, \bigsqcup)$. For the morphism part, assume that we have a function $f : A \to B$. Then $(B_\bot, \sqsubseteq, \bot, \eta \circ f, \bigsqcup)$ is an A-partiality algebra, and hence there is a morphism to this algebra from the initial A-partiality algebra $(A_\bot, \sqsubseteq, \bot, \eta, \bigsqcup)$. By removing the components $\eta \circ f : A \to B_\bot$ and $\eta : A \to A_\bot$ we get a morphism between ω-cpos.

The function η lifts to a natural transformation from the identity functor to $\mathsf{U} \circ \mathsf{F}$. In order to construct a natural transformation from $\mathsf{F} \circ \mathsf{U}$ to the identity functor, assume that we are given some ω-cpo X. We can construct an ω-cpo morphism from $\mathsf{F}(\mathsf{U}(X))$ to X by noticing that $(\mathsf{U}(X), \sqsubseteq_X, \bot_X, id, \bigsqcup_X)$ is a partiality algebra on $\mathsf{U}(X)$, and thanks to initiality we get a morphism m from $\mathsf{F}(\mathsf{U}(X))$ to X satisfying $m \circ \eta = id$. After proving some equalities we end up with the following result, where the definition of "adjoint" is taken from the HoTT book [20, Definition 9.3.1]:

Theorem 5. *For a given set A, the functor F is a left adjoint to the forgetful functor U. This means that $\mathsf{F}(A)$ can be seen as the* free *ω-cpo over A.* □

Thus we get a justification for calling the concept that we are discussing the partiality *monad*:

Corollary 6. *The composition $\mathsf{U} \circ \mathsf{F} : \mathsf{SET} \to \mathsf{SET}$, which maps objects A to $A_\bot$, is a monad.* □

Note that one can also construct a monad structure on $(-)_\bot$ directly. Let us fix the set A. The unit is given by η. For the multiplication $\mu : (A_\bot)_\bot \to A_\bot$, note that $A_\bot$ can be given the structure of a partiality algebra over $A_\bot$ in a trivial way: the underlying set is $A_\bot$, the function $\eta_{A_\bot} : A_\bot \to A_\bot$ is the identity, $\sqsubseteq_{A_\bot}$ is $\sqsubseteq$, and so on. This gives us the function μ as the unique morphism from the initial partiality algebra to this one. Proving the monad laws is straightforward.

3.3 A Characterisation of the Relation $\sqsubseteq$

To further analyse the QIIT construction, we show how the relation $\sqsubseteq$ on the set $A_\bot$ behaves.[1] These results are useful when working with the partiality monad, and will play an important role in the next section of the paper. The arguments

[1] The work presented in Sect. 3.3 was done in collaboration with Paolo Capriotti.

are only sketched here, details are given in the formalisation. We use the propositional truncation $\|-\|$ (also known as "squashing"), which turns a type into a proposition. It can be implemented by quotienting with the trivial relation.

We know that $\bot \sqsubseteq y$ is (by definition) satisfied for any $y : A_\bot$, and for the least upper bound we have that $\bigsqcup(s, q) \sqsubseteq y$ is equivalent to $\Pi_{n:\mathbb{N}}\, s_n \sqsubseteq y$. The following lemma provides a characterisation of $\eta(a) \sqsubseteq y$, for any $a : A$:

Lemma 7. *The binary relation $\sqsubseteq$ on $A_\bot$ has the following properties:*

$$\begin{aligned} \eta(a) &\sqsubseteq \bot && \leftrightarrow && \mathsf{0} \\ \eta(a) &\sqsubseteq \eta(b) && \leftrightarrow && a = b \\ \eta(a) &\sqsubseteq \bigsqcup(s, q) && \leftrightarrow && \|\Sigma_{n:\mathbb{N}}\, \eta(a) \sqsubseteq s_n\| \end{aligned}$$

We will give the proof of this lemma later and make a remark first. Constructors in "HIT-like" definitions, e.g. QIITs, may in general be neither injective nor disjoint. For instance, $\bigsqcup(\lambda n.\bot, q) = \bot$. However, we have the following lemma:

Corollary 8. *For any $a : A$ and $y : A_\bot$, we have that $\eta(a) \sqsubseteq y$ implies that $\eta(a) = y$. In particular, η is injective: if $\eta(a) = \eta(b)$, then $a = b$. Moreover, we have $\eta(a) \neq \bot$.*

Proof (of Corollary 8). The last two claims are simple consequences of the lemma and reflexivity. For the first claim, let us fix $a : A$ and apply Lemma 3 with $P(y) :\equiv \eta(a) \sqsubseteq y \to \eta(a) = y$. The only non-immediate step is the case for $\bigsqcup(s, q)$, where we can assume $\Pi_{n:\mathbb{N}}\, P(s_n)$. From $\eta(a) \sqsubseteq \bigsqcup(s, q)$ and Lemma 7 we get $\|\Sigma_{n:\mathbb{N}}\, \eta(a) \sqsubseteq s_n\|$. We are proving a proposition, so we can assume that we have $n : \mathbb{N}$ such that $\eta(a) \sqsubseteq s_n$. This implies that, for all $m \geq n$, $\eta(a) \sqsubseteq s_m$ and hence, by the "inductive hypothesis", $\eta(a) = s_m$. Thus $\eta(a)$ is an upper bound of s, so we get $\bigsqcup(s, q) \sqsubseteq \eta(a)$, which by antisymmetry implies $\bigsqcup(s, q) = \eta(a)$. □

The proof of the lemma is more technical. The approach is similar to that used to prove some results about the real numbers defined as a HIIT in the HoTT book [20, Theorems 11.3.16 and 11.3.32]. We only give a sketch here, the complete proof can be found in our Agda formalisation.

Proof (of Lemma 7). For every $a : A$ we construct a relation in $A_\bot \to \mathsf{Prop}$ by applying the elimination principle of $A_\bot$ and $\sqsubseteq$, treating Prop as a partiality algebra over A in the following way:

$$\begin{aligned} P \sqsubseteq_{\mathsf{Prop}} Q &:\equiv (P \to Q) & \eta_{\mathsf{Prop}}(b) &:\equiv (a = b) \\ \bot_{\mathsf{Prop}} &:\equiv \mathbf{0} & \bigsqcup_{\mathsf{Prop}}(S, P) &:\equiv \|\Sigma_{n:\mathbb{N}}\, S_n\| \end{aligned}$$

Propositional extensionality is used to prove that Prop is a set (this is a variant of an instance of Theorem 7.1.11 in the HoTT book [20]), and to prove the antisymmetry law.

Using Lemma 3 one can then show that the defined relation is pointwise equal to $\eta(a) \sqsubseteq -$, and it is easy to see that the relation has the properties claimed in the statement of Lemma 7. □

Using the results above one can prove that the order is flat, in the sense that if x and y are distinct from $\bot$ and $x \neq y$, then $x \not\sqsubseteq y$ (see the formalisation).

4 Relation to the Coinductive Construction

In this section we compare our QIIT to Capretta's coinductive delay monad [6], quotiented by weak bisimilarity [8]. Let us start by giving Capretta's construction, as already outlined in the introduction. $\mathcal{U}$ stands for a universe of types.

Definition 9 (delay monad and weak bisimilarity). *For a set A the* delay monad $\mathrm{D}(A)$ *is the coinductive type generated by* $\mathsf{now} : A \to \mathrm{D}(A)$ *and* $\mathsf{later} : \mathrm{D}(A) \to \mathrm{D}(A)$. *The "terminates with" relation* $\downarrow_{\mathrm{D}} : \mathrm{D}(A) \to A \to \mathcal{U}$ *is the indexed inductive type generated by two constructors of type* $\eta(a) \downarrow_{\mathrm{D}} a$ *and* $p \downarrow_{\mathrm{D}} a \to \mathsf{later}(p) \downarrow_{\mathrm{D}} a$. *Furthermore, x and $y : \mathrm{D}(A)$ are said to be* weakly bisimilar*, written $x \sim_{\mathrm{D}} y$, if $\Pi_{a:A}\, x \downarrow_{\mathrm{D}} a \leftrightarrow y \downarrow_{\mathrm{D}} a$ holds.*

It is easy to give $\mathrm{D}(A)$ the structure of a monad. Note that $x \downarrow_{\mathrm{D}} a$ can alternatively be defined to be $\Sigma_{n:\mathbb{N}}\, x = \mathsf{later}^n(\mathsf{now}(a))$. The types $x \downarrow_{\mathrm{D}} a$ and $x \sim_{\mathrm{D}} y$ are propositional, and $\sim_{\mathrm{D}}$ is an equivalence relation on $\mathrm{D}(A)$.

The goal of this section is to show that, in the presence of countable choice, the partiality monad $A_\bot$ is equivalent to $\mathrm{D}(A)/{\sim_{\mathrm{D}}}$. (We use the notion of equivalence from the HoTT book [20], which for sets is equivalent to bijective correspondence.) To understand the structure of the proof, let us observe that $\mathrm{D}(A)/{\sim_{\mathrm{D}}}$ is constructed as a "coinductive type that is quotiented afterwards", while $A_\bot$ is an "inductive type that is quotiented at the time of construction". To build a connection between these, it seems rather intuitive to consider an intermediate construction, either a "coinductive type that is quotiented at the time of construction" or an "inductive type that is quotiented afterwards". The theory of "higher coinductive types" has, as far as we know, not been explored much yet, so we go with the second option. We do not even need an *inductive* construction: it is well-known that coinductive structures can be represented using finite approximations, and here, it is enough to consider monotone functions. Thus, first we will show that $\mathrm{D}(A)$ is equivalent to a type of monotone sequences, carefully formulated, and that the equivalence lifts to the quotients. Then we will prove that, assuming countable choice, the quotiented monotone sequences are equivalent to $A_\bot$.

4.1 The Delay Monad and Monotone Sequences

For a set A we say that a function $g : \mathbb{N} \to A + \mathbf{1}$ is a monotone sequence if it satisfies the propositional property

$$\mathsf{ismon}(g) \;:\equiv\; \Pi_{n:\mathbb{N}}\, (g_n = g_{n+1}) + ((g_n = \mathsf{inr}(\star)) \times (g_{n+1} \neq \mathsf{inr}(\star)))\,.$$

The set of monotone sequences, $\Sigma_{g:\mathbb{N}\to A+\mathbf{1}}\, \mathsf{ismon}(g)$, is denoted by Seq_A. Below the notation $-_n$ will be used not only for functions, but also for monotone sequences; $(g, p)_n$ means g_n.

As Chapman et al. [8] observe, one can construct a sequence of type $\mathbb{N} \to A+\mathbf{1}$ from an element of $\mathrm{D}(A)$. If their construction is tweaked a little, then the resulting sequences are monotone, and the map is an equivalence:

Lemma 10. *The types* Seq_A *and* $\mathrm{D}(A)$ *are equivalent.*

Proof. We can simply give functions back and forth. Note that endofunctions on $\mathrm{D}(A)$ that correspond to later and "remove later, if there is one" can be mimicked for Seq_A: let us use the names *shift* and *unshift* : $\mathsf{Seq}_A \to \mathsf{Seq}_A$ for the functions that are determined by $\mathit{shift}(g)_0 :\equiv \mathsf{inr}(\star)$, $\mathit{shift}(g)_{n+1} :\equiv g_n$, and $\mathit{unshift}(g)_n :\equiv g_{n+1}$.

Define $j : \mathrm{D}(A) \to \mathsf{Seq}_A$ such that $j(\mathsf{now}(a))$ equals $\lambda n.\mathsf{inl}(a)$, and $j(\mathsf{later}(x))$ equals $\mathit{shift}(j(x))$, One way to do this is to define $j(z)_n$ by recursion on n, followed by case distinction on z. Furthermore, define $h : \mathsf{Seq}_A \to \mathrm{D}(A)$ in the following way: Given $s : \mathsf{Seq}_A$, do case distinction on s_0. If s_0 is $\mathsf{inl}(a)$, return $\mathsf{now}(a)$. Otherwise, return $\mathsf{later}(h(\mathit{unshift}(s)))$. It is straightforward to show that j and h are inverses of each other. □

As an aside, our formalisation shows that Lemma 10 holds even if A is not a set.

Next, we mimic the relation $\downarrow_{\mathrm{D}}$ by setting

$$\begin{aligned} &\downarrow_{\mathsf{Seq}} : \mathsf{Seq}_A \to A \to \mathcal{U} \\ &s \downarrow_{\mathsf{Seq}} a \;:\equiv\; \Sigma_{n:\mathbb{N}}\, s_n = \mathsf{inl}(a). \end{aligned}$$

The relation $\downarrow_{\mathsf{Seq}}$ is not in general propositional. To remedy this, we can truncate and consider $\|s \downarrow_{\mathsf{Seq}} a\|$. Using strategies explained by Kraus et al. [15], we have $\|s \downarrow_{\mathsf{Seq}} a\| \to s \downarrow_{\mathsf{Seq}} a$, so we can always extract a concrete value of n: a variant of the definition above in which the number n is required to be minimal is propositional, and this definition can be shown to be logically equivalent to both $\|s \downarrow_{\mathsf{Seq}} a\|$ and $s \downarrow_{\mathsf{Seq}} a$. See the formalisation for details.

We define the propositional relations $\sqsubseteq_{\mathsf{Seq}}$ and $\sim_{\mathsf{Seq}}$ by

$$\begin{aligned} s \sqsubseteq_{\mathsf{Seq}} t \;&:\equiv\; \Pi_{a:A}\, \|s \downarrow_{\mathsf{Seq}} a\| \to \|t \downarrow_{\mathsf{Seq}} a\| \text{ and} \\ s \sim_{\mathsf{Seq}} t \;&:\equiv\; s \sqsubseteq_{\mathsf{Seq}} t \times t \sqsubseteq_{\mathsf{Seq}} s. \end{aligned}$$

By checking that the equivalence from Lemma 10 maps $\sim_{\mathsf{Seq}}$-related elements to $\sim_{\mathrm{D}}$-related elements, we get:

Lemma 11. *The sets* $\mathsf{Seq}_A/{\sim_{\mathsf{Seq}}}$ *and* $\mathrm{D}(A)/{\sim_{\mathrm{D}}}$ *are equivalent.* □

4.2 Monotone Sequences and the QIIT Construction

As the final step of showing that $\mathrm{D}(A)/{\sim_{\mathrm{D}}}$ and $A_\perp$ are equivalent, we show that $\mathsf{Seq}_A/{\sim_{\mathsf{Seq}}}$ and $A_\perp$ are. The plan is as follows: There is a canonical function $w : \mathsf{Seq}_A \to A_\perp$ which can be extended to a function $\tilde{w} : \mathsf{Seq}_A/{\sim_{\mathsf{Seq}}} \to A_\perp$. The function $\tilde{w}$ is injective. Furthermore, if we assume countable choice, then the function w, and thus also $\tilde{w}$, are surjective. Thus $\tilde{w}$ is an equivalence.

Let us start by constructing w and $\tilde{w}$. We use a copairing function $[\eta \mid \bot] : A + \mathbf{1} \to A_\bot$ defined by $[\eta \mid \bot]\,(\mathsf{inl}(a)) :\equiv \eta(a)$ and $[\eta \mid \bot]\,(\mathsf{inr}(\star)) :\equiv \bot$, and define $w : \mathsf{Seq}_A \to A_\bot$ by $w(s,q) :\equiv \bigsqcup([\eta \mid \bot] \circ s, \ldots)$, with a canonical proof of monotonicity.

Lemma 12. *The function w is monotone: $\Pi_{s,t:\mathsf{Seq}_A}\, s \sqsubseteq_{\mathsf{Seq}} t \to w(s) \sqsubseteq w(t)$. Thus w extends to a map $\tilde{w} : \mathsf{Seq}_A/{\sim_{\mathsf{Seq}}} \to A_\bot$.*

Proof. For the second claim we show that w maps elements related by $\sim_{\mathsf{Seq}}$ to equal elements. This follows from the first claim by antisymmetry. For the first claim it suffices to find a function $k : \mathbb{N} \to \mathbb{N}$ such that, for all n, we have $[\eta \mid \bot]\,(s_n) \sqsubseteq [\eta \mid \bot]\,(t_{k(n)})$. Fix n. If s_n is $\mathsf{inr}(\star)$, then $[\eta \mid \bot]\,(s_n)$ is $\bot$, and $k(n)$ can thus be chosen arbitrarily. If s_n is $\mathsf{inl}(a)$, then we have $s \downarrow_{\mathsf{Seq}} a$ and therefore $t \downarrow_{\mathsf{Seq}} a$, which gives us a number $k(n)$ such that $t_{k(n)} = \mathsf{inl}(a)$ and $[\eta \mid \bot]\,(s_n) = \eta(a) = [\eta \mid \bot]\,(t_{k(n)})$. □

Lemma 13. *The function $\tilde{w}$ is injective: $\Pi_{s,t:\mathsf{Seq}_A/{\sim_{\mathsf{Seq}}}}\, \tilde{w}(s) = \tilde{w}(t) \to s = t$.*

Proof. It suffices to show that, for $s, t : \mathsf{Seq}_A$, $w(s) = w(t)$ implies $s \sim_{\mathsf{Seq}} t$. By symmetry, it is enough to fix $a : A$ and show $\|s \downarrow_{\mathsf{Seq}} a\| \to \|t \downarrow_{\mathsf{Seq}} a\|$, which follows from $s \downarrow_{\mathsf{Seq}} a \to \|t \downarrow_{\mathsf{Seq}} a\|$. If $s \downarrow_{\mathsf{Seq}} a$ then $w(s) = \eta(a)$, and thus also $w(t) = \eta(a)$. Using Lemma 7 and Corollary 8 we then get $\|\Sigma_{n:\mathbb{N}}\, \eta(a) = [\eta \mid \bot]\,(t_n)\|$, which implies $\|\Sigma_{n:\mathbb{N}}\, t_n = \mathsf{inl}(a)\|$. □

Lemma 14. *Under countable choice, w is surjective: $\Pi_{x:A_\bot}\, \|\Sigma_{s:\mathsf{Seq}_A}\, w(s) = x\|$.*

Proof. We apply the simplified induction principle presented in Lemma 3. The propositional predicate is $P(x) :\equiv \|\Sigma_{s:\mathsf{Seq}_A}\, w(s) = x\|$. Both $P(\bot)$ and $P(\eta(a))$ are trivial: in the first case we use the sequence that is constantly $\mathsf{inr}(\star)$, while in the second case we take the one that is constantly $\mathsf{inl}(a)$.

The interesting part is to show $P\,(\bigsqcup(f,p))$ for a given $f : \mathbb{N} \to A_\bot$ and $p : \Pi_{n:\mathbb{N}}\, f_n \sqsubseteq f_{n+1}$. By the mentioned induction principle, we can assume $\Pi_{n:\mathbb{N}}\, P(f_n)$, which unfolds to $\Pi_{n:\mathbb{N}}\, \|\Sigma_{t:\mathsf{Seq}_A}\, w(t) = f_n\|$. By countable choice, we can swap $\Pi_{n:\mathbb{N}}$ and $\|-\|$, which allows us to remove the truncation completely, because the goal is propositional. Hence we can assume $\Pi_{n:\mathbb{N}}\, \Sigma_{t:\mathsf{Seq}_A}\, w(t) = f_n$.

Using the usual distributivity law for Π and Σ (sometimes called the "type-theoretic axiom of choice"), we can assume that we are given $g : \mathbb{N} \to \mathsf{Seq}_A$ and a proof $\gamma : \Pi_{n:\mathbb{N}}\, w(g_n) = f_n$. By dropping the monotonicity proof and uncurrying, g gives us a function $g' : \mathbb{N} \times \mathbb{N} \to A + \mathbf{1}$ with the property that it assumes at most one value in A: If $g'_{i,j} = \mathsf{inl}(a)$, then (using γ) $\eta(a) \sqsubseteq f_i$, thus $\eta(a) \sqsubseteq \bigsqcup(f,p)$, and hence $\bigsqcup(f,p) = \eta(a)$ by Corollary 8. If we also have $g'_{k,m} = \mathsf{inl}(b)$, then $\eta(a) = \eta(b)$, which by Corollary 8 implies that $a = b$.

We use g' to construct an element of Seq_A. Take an arbitrary isomorphism $\sigma : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ (a split surjection would also be sufficient), and define a function $\tilde{g} : \mathbb{N} \to A + \mathbf{1}$ by

$$\tilde{g}(n) :\equiv \begin{cases} g'(\sigma_n), & \text{if } n = 0 \text{ or } g'(\sigma_n) \neq \mathsf{inr}(\star), \\ \tilde{g}(n-1), & \text{otherwise.} \end{cases}$$

The intuition is that $\tilde{g}(n)$ checks the first $n+1$ results of $g' \circ \sigma$ and chooses the last which is of the form $\mathsf{inl}(-)$, if any, otherwise returning $\mathsf{inr}(\star)$. Because g' assumes at most one value in A we get that $\tilde{g}$ is monotone, $q : \mathsf{ismon}(\tilde{g})$. Furthermore $(\tilde{g}, q) \downarrow_{\mathsf{Seq}} a$ holds if and only if we have $\Sigma_{n:\mathbb{N}}\, g'(\sigma_n) = \mathsf{inl}(a)$.

In order to complete the proof of $P\left(\bigsqcup(f,p)\right)$ we show that $w(\tilde{g}, q) = \bigsqcup(f,p)$ by using antisymmetry:

– *First part:* $w(\tilde{g}, q) \sqsubseteq \bigsqcup(f,p)$. After unfolding the definition of w we see that it suffices to prove $[\eta \mid \bot]\,(\tilde{g}_n) \sqsubseteq \bigsqcup(f,p)$ for an arbitrary $n : \mathbb{N}$. If $\tilde{g}_n$ is $\mathsf{inr}(\star)$, then this is trivial. If $\tilde{g}_n$ is $\mathsf{inl}(a)$ for some $a : A$, then we can find a pair (i,j) such that $g'_{i,j} = \mathsf{inl}(a)$. Thus we get the following chain:

$$[\eta \mid \bot]\,(\tilde{g}_n) \;=\; [\eta \mid \bot]\,(g'_{i,j}) \;\sqsubseteq\; w(g_i) \;=\; f_i \;\sqsubseteq\; \bigsqcup(f,p)$$

– *Second part:* $\bigsqcup(f,p) \sqsubseteq w(\tilde{g}, q)$. Given $n : \mathbb{N}$, we show that $f_n \sqsubseteq w(\tilde{g}, q)$. By γ_n we have $f_n = w(g_n)$. Thus it suffices to prove $w(g_n) \sqsubseteq w(\tilde{g}, q)$, which by Lemma 12 follows if $g_n \sqsubseteq_{\mathsf{Seq}} (\tilde{g}, q)$. If $g_n(i) = \mathsf{inl}(a)$ for some i and a, then we have $\mathsf{inl}(a) = g'_{n,i} = g'(\sigma(\sigma^{-1}_{n,i}))$, and thus $(\tilde{g}, q) \downarrow_{\mathsf{Seq}} a$. □

This immediately shows that $\tilde{w}$ is surjective as well. Putting the pieces together, we get the main result of this section:

Theorem 15. *In the presence of countable choice the map $\tilde{w}$ is an equivalence. Hence the three sets $\mathrm{D}(A)/{\sim_{\mathrm{D}}}$, $\mathsf{Seq}_A/{\sim_{\mathsf{Seq}}}$ and $A_\bot$ are equivalent.*

Proof. A function between sets is an equivalence exactly if it is surjective and injective. This is a special case of Theorem 4.6.3 in the HoTT book [20], which states that a function between arbitrary types is an equivalence if and only if it is surjective and an *embedding*, which for sets is equivalent to being injective. □

5 Applications

The following examples show that our construction can be used in formalisations.

5.1 Nonterminating Functions as Fixed Points

Partiality algebras can be used to implement not necessarily terminating functions. Let $(Y, \sqsubseteq_Y, \bot_Y, \eta_Y, \bigsqcup_Y)$ be a partiality algebra, and let $\varphi : Y \to Y$ be a monotone and ω-continuous function. We can write down the least fixed point of φ directly as $\bigsqcup_Y(\lambda n.\varphi^n(\bot_Y), p)$, where p is constructed from the fact that $\bot_Y \sqsubseteq_Y \varphi(\bot_Y)$ and from the monotonicity proof of φ. One does not need ω-continuity to write down this expression, but we use it to prove that the expression is a fixed point of φ.

If $(Y, \sqsubseteq_Y, \bot_Y, \eta_Y, \bigsqcup_Y)$ is a partiality algebra and X is any type, then the function space $X \to Y$ can be given the structure of a partiality algebra in a canonical way (this is done for *dependent* types $\Pi_{x:X}\, Y(x)$ in the formalisation).

As an example of how this kind of partiality algebra can be used we will construct a function $search_q : A^\omega \to A_\perp$ that takes an element of the coinductive set of streams A^ω and searches for an element of the set A satisfying the decidable predicate $q : A \to \mathbf{2}$. The function is constructed as the least fixed point of the following endofunction on $A^\omega \to A_\perp$:

$$\Phi(f)(a :: as) :\equiv \mathsf{if}\ q(a)\ \mathsf{then}\ \eta(a)\ \mathsf{else}\ f(as)$$

It is straightforward to check that $f \sqsubseteq g$ implies $\Phi(f) \sqsubseteq \Phi(g)$ by applying $\Phi(f)$ and $\Phi(g)$ to a point $a :: as$ and doing case analysis on $q(a)$. Thus Φ is monotone. In a similar way one can verify that Φ is ω-continuous.

5.2 Functions from the Reals

Let us consider the Cauchy reals, defined as a quotient. We say that $f : \mathbb{N} \to \mathbb{Q}$ is a *Cauchy sequence* if, for all $m, n : \mathbb{N}$ with $m < n$, we have $-1 < m \cdot (f_m - f_n) < 1$. Furthermore, f and g are equivalent (written $f \sim g$) if, for all $n : \mathbb{N}$, we have $-2 \leq n \cdot (f_n - g_n) \leq 2$. We use the notation $\mathbb{R}^q$ for the quotient of Cauchy sequences by $\sim$.

A meta-theoretic result is that, without further assumptions, any definable function (i.e. any closed term) of type $\mathbb{R}^q \to \mathbf{2}$ is constant for reasons of continuity [16]. In particular, we cannot define a function *isPositive* which checks whether a real number is positive. However, we *can* define a function *isPositive* : $\mathbb{R}^q \to \mathbf{2}_\perp$ such that *isPositive*(r) is equal—but not judgmentally/definitionally equal—to $\eta(1_\mathbf{2})$ if r is positive, $\eta(0_\mathbf{2})$ if r is negative, and $\perp$ if r is zero.

We define this function as follows: Given a Cauchy sequence $f : \mathbb{N} \to \mathbb{Q}$, we construct a new sequence $\overline{f} : \mathbb{N} \to \{-, ?, +\}$. The idea is that $\overline{f}_n$ is an approximation which only takes f_i with $i \leq n$ into account. We start with $\overline{f}_0 :\equiv ?$. If we have chosen $\overline{f}_{n-1}$ to be $-$, then we choose $\overline{f}_n$ to be $-$ as well, and analogously for $+$. If we have chosen $\overline{f}_{n-1}$ to be $?$, we check whether $f_n \cdot n < -2$, in which case we choose $\overline{f}_n$ to be $-$; if $f_n \cdot n > 2$, we choose $\overline{f}_n$ to be $+$; otherwise, we choose $\overline{f}_n$ to be $?$. We can compose with the map $\{-, ?, +\} \to A_\perp$ which is defined by $- \mapsto \eta(0_\mathbf{2})$, $? \mapsto \perp$, and $+ \mapsto \eta(1_\mathbf{2})$. This defines a monotone sequence in $\mathbf{2}_\perp$, and we can form $\bigsqcup \overline{f} : \mathbf{2}_\perp$ to answer whether f represents a positive or negative number, or is zero. One can check that equivalent Cauchy sequences get mapped to equal values, hence we get *isPositive* : $\mathbb{R}^q \to \mathbf{2}_\perp$.

The strategy outlined above does not quite work for the reals defined as a HIIT [20] because, roughly speaking, in that setting f_n is a real number and a comparison such as $f_n \cdot n < -2$ is undecidable. Recently Gilbert has refined our approach and defined a function *isPositive* for such reals [13], using the definition of the partiality monad presented in this text (with insignificant differences). Gilbert's key observation is that comparisons between real numbers and rational numbers are semidecidable, and semidecidability is sufficient to define *isPositive*.

5.3 Operational Semantics

In previous work the second-named author has discussed how one can use the delay monad to express operational semantics as definitional interpreters [11]. As a case study we have ported some parts of this work to the partiality monad discussed in the present text: definitional interpreters for a simple functional language and a simple virtual machine, a type soundness proof, a compiler, and a compiler correctness result. Due to lack of space we do not include any details here, but refer interested readers to the accompanying source code.

6 Discussion and Further Work

We have constructed a partiality monad without using countable choice. This is only a first step in the development of a form of constructive domain theory in type theory. It remains to be seen whether it is possible to, for instance, replicate the work of Benton et al. [5], who develop domain theory using the delay monad.

Consider the partial function $\mathit{filter} : \Pi_{A:\mathsf{Set}}\,(A \to \mathbf{2}) \to A^\omega \to A^\omega$ that filters out elements from a stream. How should partial streams over A be defined? Defining them as $\nu X.(A \times X)_\perp$ seems inadequate, because the ordering of $(-)_\perp$ is flat. One approach would perhaps be to define this type by solving a domain equation. Instead of relying on the type-theoretic mechanism to define recursive types, we can perhaps construct a suitable type of partial streams as the colimit of an ω-cocontinuous functor on the category of ω-cpos. Preliminary investigations indicate that QIITs are useful in the endeavour, for example in the definition of a lifting comonad on ω-cpos (as suggested by Paolo Capriotti).

Going in another direction, it might be worth investigating how much topology can be done using the Sierpinski space, represented as $\mathbf{1}_\perp$ in our setting. A very similar question was discussed at the Special Year on Univalent Foundations of Mathematics at the IAS in Princeton (2012–2013). Moreover, some observations have been presented by Gilbert [13], who used our definition of $\mathbf{1}_\perp$ as presented in this paper (with minor differences).

Acknowledgements. We thank Gershom Bazerman, Paolo Capriotti, Bernhard Reus, and Bas Spitters for interesting discussions and pointers to related work, and the anonymous reviewers for useful feedback. The work presented in Sect. 3.3 was done in collaboration with Paolo Capriotti.

References

1. Altenkirch, T., Kaposi, A.: Type theory in type theory using quotient inductive types. In: Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, pp. 18–29 (2016). doi:10.1145/2837614.2837638
2. Altenkirch, T., Capriotti, P., Dijkstra, G., Nordvall Forsberg, F.: Quotient inductive-inductive types. Preprint arXiv:1612.02346v1 [cs.LO] (2016)

3. Altenkirch, T., Danielsson, N.A., Kraus, N.: Code related to the paper "Partiality, revisited: The partiality monad as a quotient inductive-inductive type" (2017). Agda code, http://www.cse.chalmers.se/~nad/
4. Awodey, S., Gambino, N., Sojakova, K.: Inductive types in homotopy type theory. In: 2012 27th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), pp. 95–104 (2012). doi:10.1109/LICS.2012.21
5. Benton, N., Kennedy, A., Varming, C.: Some domain theory and denotational semantics in coq. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs 2009. LNCS, vol. 5674, pp. 115–130. Springer, Heidelberg (2009). doi:10.1007/978-3-642-03359-9_10
6. Capretta, V.: General recursion via coinductive types. Logical Methods Comput. Sci. **1**(2), 1–28 (2005). doi:10.2168/LMCS-1(2:1)2005
7. Capretta, V., Altenkirch, T., Uustalu, T.: Partiality is an effect. Slides for a talk given by Uustalu at the 22nd Meeting of IFIP Working Group 2.8 (2005). http://www.cs.ox.ac.uk/ralf.hinze/WG2.8/22/slides/tarmo.pdf
8. Chapman, J., Uustalu, T., Veltri, N.: Quotienting the delay monad by weak bisimilarity. In: Leucker, M., Rueda, C., Valencia, F.D. (eds.) ICTAC 2015. LNCS, vol. 9399, pp. 110–125. Springer, Cham (2015). doi:10.1007/978-3-319-25150-9_8
9. Cockx, J., Abel, A.: Sprinkles of extensionality for your vanilla type theory. In: TYPES 2016, Types for Proofs and Programs, 22nd Meeting, Book of Abstracts (2016). http://www.types2016.uns.ac.rs/images/abstracts/cockx.pdf
10. Coquand, T., Mannaa, B., Ruch, F.: Stack semantics of type theory. Preprint arXiv:1701.02571v1 [cs.LO] (2017)
11. Danielsson, N.A.: Operational semantics using the partiality monad. In: Proceedings of the 17th ACM SIGPLAN International Conference on Functional Programming, ICFP 2012, pp. 127–138 (2012). doi:10.1145/2364527.2364546
12. Dijkstra, G.: Quotient inductive-inductive definitions. Ph.D. thesis, University of Nottingham (2017). In preparation
13. Gilbert, G.: Formalising real numbers in homotopy type theory. In: Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2017, pp. 112–124 (2017). doi:10.1145/3018610.3018614
14. Hofmann, M.: Extensional concepts in intensional type theory. Ph.D. thesis, University of Edinburgh (1995)
15. Kraus, N., Escardó, M., Coquand, T., Altenkirch, T.: Generalizations of Hedberg's theorem. In: Hasegawa, M. (ed.) TLCA 2013. LNCS, vol. 7941, pp. 173–188. Springer, Heidelberg (2013). doi:10.1007/978-3-642-38946-7_14
16. Li, N.: Quotient types in type theory. Ph.D. thesis, University of Nottingham (2015)
17. Lubarsky, R.S.: On the Cauchy completeness of the constructive Cauchy reals. Math. Logic Quart. **53**(4–5), 396–414 (2007). doi:10.1002/malq.200710007
18. Richman, F.: Constructive mathematics without choice. In: Schuster, P., Berger, U., Osswald, H. (eds.) Reuniting the Antipodes – Constructive and Nonstandard Views of the Continuum. Synthese Library, vol. 306, pp. 199–205. Springer Science+Business Media, Dordrecht (2001). doi:10.1007/978-94-015-9757-9_17
19. Sojakova, K.: Higher inductive types as homotopy-initial algebras. In: Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, pp. 31–42 (2015). doi:10.1145/2676726.2676983
20. The Univalent Foundations Program: Homotopy Type Theory: Univalent Foundations of Mathematics, 1st edn. (2013). https://homotopytypetheory.org/book/

On the Semantics of Intensionality

G.A. Kavvos(✉)

Department of Computer Science, University of Oxford, Wolfson Building, Parks Road, Oxford OX1 3QD, UK
alex.kavvos@cs.ox.ac.uk

Abstract. In this paper we propose a categorical theory of intensionality. We first revisit the notion of intensionality, and discuss its relevance to logic and computer science. It turns out that 1-category theory is not the most appropriate vehicle for studying the interplay of extension and intension. We are thus led to consider the P-categories of Čubrić, Dybjer and Scott, which are categories only up to a partial equivalence relation (PER). In this setting, we introduce a new P-categorical construct, that of exposures. Exposures are very nearly functors, except that they do not preserve the PERs of the P-category. Inspired by the categorical semantics of modal logic, we begin to develop their theory. Our leading examples demonstrate that an exposure is an abstraction of well-behaved intensional devices, such as Gödel numberings. The outcome is a unifying framework in which classic results of Kleene, Gödel, Tarski and Rice find concise, clear formulations, and where each logical device or assumption involved in their proofs can be expressed in the same algebraic manner.

1 Introduction: Intensionality and Intensional Recursion

This paper proposes a new theory of *intensionality*. Intensionality is a notion that dates as early as Frege's philosophical distinction between *sense* and *denotation*, see for example [11]. In the most mathematically general sense, to be 'intensional' is to somehow operate at a level finer than some predetermined 'extensional' equality. Whereas in mainstream mathematics intensionality is merely a nuisance, it is omnipresent in computer science, where the objects of study are distinct programs and processes describing (often identical) abstract mathematical values. At one end of the spectrum, programs are extensionally equal if they are *observationally equivalent*, i.e. interchangeable in any context. At the other extreme, computer viruses often make decisions simply on patterns of object code they encounter, disregarding the actual function of what they are infecting; one could say they operate up to syntactic identity.

1.1 Intensionality as a Logical Construct

We are interested in devising a categorical setting in which programs can be viewed in two ways simultaneously, either as *black boxes*—i.e. extensionally,

This work was supported by the EPSRC (award reference 1354534).

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 550–566, 2017.
DOI: 10.1007/978-3-662-54458-7_32

whatever we define that to mean, but also as *white boxes*—i.e. intensionally, which should amount to being able to 'look inside' a construction and examine its internal workings.

There are many reasons for pursuing this avenue. The main new construct we will introduce will be an abstraction of the notion of *Gödel numbering*. The immediate achievement of this paper is a categorical language in which we can state many classic theorems from logic and computability that depend on the interplay between extension and intension. This unifying language encompasses all such 'diagonal constructions' in a way that makes the ingredients involved in each argument clear. As such, we regard this as an improvement on the classic paper of Lawvere [16].

A more medium-term goal is the quest to prove a logical foundation to *computational reflection*, in the sense of Brian Cantwell Smith [23]. A reflective program is always able to obtain a complete description of its source code and current state; this allows it to make decisions depending on both its syntax and runtime behaviour. This strand of research quickly ran into impossibility results that demonstrate that reflective features are logically ill-behaved: see e.g. [2] for reflection in untyped λ-calculus, or [25] for a more involved example involving the LISP **fexpr** construct. Our viewpoint allows us to talk about the notion of *intensional recursion*, which is more general than ordinary extensional recursion, and seems to correspond to a well-behaved form of reflection. We connect this to a classic result in computability theory, namely Kleene's *Second Recursion Theorem (SRT)*.

Finally, a more long-term goal is to understand *non-functional computation*. In this context, non-functional computation means something much more general than just computing with side-effects: we are interested in general higher-order computation acting 'on syntax.' Approaches to such forms of computation have hitherto been ad-hoc, see e.g. [18, Sect. 6]. We would like to provide a very general way to add 'intensional' features to a pure functional programming language.

1.2 Prospectus

To begin, we introduce in Sect. 2 a known connection between the notion of intension and the necessity modality from modal logic, and the use of *modal types* in isolating intension from extension. We argue that this connection cannot be fully substantiated in 1-category theory. Hence, we introduce *P-categories* and explain their use in modelling intensionality.

In Sect. 3 we introduce a new P-categorical construct, the *exposure*. Exposures 'turn intensions into extensions' in a manner inspired by the modality-as-intension interpretation. In Sect. 4 we use exposures to talk about the notion of intensional recursion in terms of *intensional fixed points (IFPs)*.

In Sect. 5 we use IFPs to prove abstract analogues to *Tarski's undefinability theorem* and Gödel's *First Incompleteness Theorem*. In Sect. 6 we construct a P-category and an endoexposure that substantiate the claim that the abstract versions correspond to the usual theorems.

We then ask the obvious question: where do IFPs come from? In Sect. 7 we generalize Lawvere's fixed-point theorem to yield IFPs. Then, in Sect. 8, we draw a parallel between Kleene's *First Recursion Theorem (FRT)* and Lawvere's fixed point result, whereas we connect our IFP-yielding result with Kleene's *Second Recursion Theorem*. We substantiate this claim in Sect. 9 by constructing a P-category and exposure based on realizability theory.

Finally, in Sect. 10 we provide further evidence for the usefulness of our language by reproducing an abstract version of *Rice's theorem*, a classic result in computability theory. We find that this is already substantiated in the P-category constructed in Sect. 9.

2 Modality-as-Intension

All the negative results regarding intensionality and computational reflection have something in common: they invariably apply to some construct that can *turn extension into intension*. For example, in [2] a contradiction is derived from the assumption that some term Q satisfies $QM =_\beta \ulcorner M \urcorner$ for any M, where the RHS is a Gödel number of M. The moral is that *one should not mix intension and extension*.

To separate the two, we will use *modal types*, as first suggested by Davies and Pfenning [10,22]. In *op. cit.* the authors use modal types to simulate two-level λ-calculi. In passing, they interpret the modal type $\Box A$ as the type of 'intensions of type A' or 'code of type A.' The T axiom $\Box A \to A$ may then be read as an *interpreter* that maps code to values, whereas the $\mathsf{4}$ axiom $\Box A \to \Box\Box A$ corresponds to *quoting*, but quoting that can only happen when the initial value is already code, and not a 'runtime,' live value.

Unfortunately, the available semantics do not corroborate this interpretation. The categorical semantics of the $\mathsf{S4}$ necessity modality—due to Bierman and de Paiva [4]—specify that $\Box$ is a *monoidal comonad* $(\Box, \epsilon, \delta)$ on a CCC. The problem is now rather obvious, in that equality in the category *is* extensional equality: if $f = g$, then $\Box f = \Box g$. In modal type theory, this amounts to saying that $\vdash M = N : A$ implies $\vdash \mathsf{box}\, M = \mathsf{box}\, N : \Box A$, which is not what we mean by intensionality at all. By definition, 'intension' should not be preserved under equality, and in [10] it is clearly stated that there should be no reductions under a $\mathsf{box}\ (-)$ construct.

To salvage this *modality-as-intension* interpretation, we have to leave 1-category theory, and move to a framework where we can separate *extensional equality*—denoted $\sim$—and *intensional equality*—denoted $\approx$. We will thus use the *P-categories* of Čubrić, Dybjer and Scott [8]. P-categories are only categories up to a family of partial equivalence relations (PERs). In our setting, the PER will specify extensional equality. All that remains is to devise a construct that (a) behaves like a modality, so that intension and extension stay separate, and (b) unpacks the non-extensional features of an arrow. This will be the starting point of our theory of *exposures*.

2.1 P-categories and Intensionality

Suppose we have a model of computation or a programming language whose programs are seen as computing functions, and suppose that we are able to *compose* programs in this language, so that given programs $\mathtt{P}$ (computing f) and $\mathtt{Q}$ (computing g) there is a simple syntactic construction $\mathtt{Q};\mathtt{P}$ (computing $g \circ f$). In more elegant cases, like the λ-calculus, composition will be substitution of a term for a free variable. But in most other cases there will be unappealing overhead, involving e.g. some horrible disjoint unions of sets of states. This syntactic overhead almost always ensures that composition of programs is not associative: $(\mathtt{R};\mathtt{Q});\mathtt{P}$ is *not* syntactically identical to $\mathtt{R};(\mathtt{Q};\mathtt{P})$, even though they compute the same function.

To model this we will use *P-categories*, first introduced in [8]. Generally denoted $\mathfrak{B}, \mathfrak{C}, \ldots$, or even $(\mathfrak{B}, \sim)$, P-categories are categories whose hom-sets are not sets, but *P-sets*: a P-set is a pair $A = (|A|, \sim_A)$ of a set $|A|$ and a partial equivalence relation (PER)[1] on $|A|$. If $a \sim_A a'$, then a can be thought of as 'equal' to a'. The lack of reflexivity means that there may be some $a \in |A|$ such that $a \not\sim_A a$: these can be thought of as points which are *not well-defined*. A *P-function* $f : A \to B$ between two P-sets $A = (|A|, \sim_A)$ and $B = (|B|, \sim_B)$ is a function $|f| : |A| \to |B|$ that respects the PER: if $a \sim_A a'$ then $|f|(a) \sim_B |f|(a')$. We simply write $f(a)$ if $a \sim_A a$.

Thus, we take each hom-set $\mathfrak{C}(A, B)$ of a P-category $\mathfrak{C}$ to be a P-set. We will only write $f : A \to B$ if $f \sim_{\mathfrak{C}(A,B)} f$, i.e. f is a well-defined arrow. Arrows in a P-category are *intensional constructions*. Two arrows $f, g : A \to B$ will be *extensionally equal* if $f \sim_{\mathfrak{C}(A,B)} g$. The axioms of category theory will then only hold up to the family of PERs, i.e. $\sim = \{\sim_{\mathfrak{C}(A,B)}\}_{A,B \in \mathfrak{C}}$. For example, it may be that $f \circ (g \circ h) \neq (f \circ g) \circ h$, yet $f \circ (g \circ h) \sim (f \circ g) \circ h$.

Hence, we regain all the standard equations of 1-categories, up to PERs. Furthermore, the standard notions of terminal objects, products and exponentials all have a P-variant in which the defining equations hold up to $\sim$. We are unable to expound on P-categories any further, but please refer to [8] for more details.

3 Exposures

We can now formulate the definition of exposures. An exposure is *almost* a (P-)functor: it preserves identity and compositions, but it only *reflects* PERs.

Definition 1. *An exposure* $Q : (\mathfrak{B}, \sim) \nrightarrow (\mathfrak{C}, \sim)$ *consists of (a) an object* $QA \in \mathfrak{C}$ *for each object* $A \in \mathfrak{B}$*, and (b) an arrow* $Qf : QA \to QB$ *in* $\mathfrak{C}$ *for each arrow* $f : A \to B$ *in* $\mathfrak{B}$*, such that (1)* $Q(id_A) \sim id_{QA}$*, and (2)* $Q(g \circ f) \sim Qg \circ Qf$ *for any arrows* $f : A \to B$ *and* $g : B \to C$*, and (3) for any* $f, g : A \to B$*, if* $Qf \sim Qg$ *then* $f \sim g$*.*

[1] That is, a symmetric and transitive relation.

The *identity exposure* $\mathsf{Id}_{\mathfrak{B}} : \mathfrak{B} \nrightarrow \mathfrak{B}$ maps every object to itself, and every arrow to itself. Finally, it is easy to see that the composite of two exposures is an exposure.

As exposures give a handle on the internal structure of arrows, they can be used to define intensional equality: if the images of two arrows under the same exposure Q are extensionally equal, then the arrows have the same implementation, so they are intensionally equal. This is an exact interpretation of a slogan of Abramsky [1]: *intensions become extensions*.

Definition 2 (Intensional Equality). *Let there be P-categories $\mathcal{B}$, $\mathcal{C}$, and an exposure $Q : (\mathcal{B}, \sim) \nrightarrow (\mathcal{C}, \sim)$. Two arrows $f, g : A \to B$ are* intensionally equal (up to Q)*, written $f \approx g$, just if $Qf \sim Qg$.*

It is obvious then that the last axiom on the definition of exposures means that intensional equality implies extensional equality.

To re-interpret concepts from the modality-as-intension interpretation—such as interpreters, quoting etc.—we shall need a notion of transformation between exposures.

Definition 3. *A* natural transformation of exposures $t : F \overset{\bullet}{\nrightarrow} G$ *where $F, G : \mathfrak{B} \nrightarrow \mathfrak{C}$ are exposures, consists of an arrow $t_A : FA \to GA$ of $\mathfrak{C}$ for each object $A \in \mathfrak{B}$, such that, for every arrow $f : A \to B$ of $\mathfrak{B}$, the following diagram commutes up to $\sim$:*

$$\begin{array}{ccc} FA & \xrightarrow{Ff} & FB \\ {\scriptstyle t_A}\downarrow & & \downarrow{\scriptstyle t_B} \\ GA & \xrightarrow[Gf]{} & GB \end{array}$$

3.1 Cartesian Exposures

Bare exposures offer no promises or guarantees regarding intensional equality. For example, it is not a given that $\pi_1 \circ \langle f, g \rangle \approx f$. However, one may argue that this equality should hold, insofar as there is no grand intensional content in projecting a component. This leads to the following notion:

Definition 4. *A exposure $Q : \mathfrak{B} \nrightarrow \mathfrak{C}$ where $\mathfrak{B}$ is a cartesian P-category is itself* cartesian *just if, for arrows $f : C \to A$ and $g : C \to B$, we have*

$$\pi_1 \circ \langle f, g \rangle \approx f, \quad \pi_2 \circ \langle f, g \rangle \approx g, \quad \text{and } \langle \pi_1 \circ h, \pi_2 \circ h \rangle \approx h$$

However, this is not enough to formally regain standard equations like $\langle f, g \rangle \circ h \approx \langle f \circ h, g \circ h \rangle$. We need to also require that exposures 'extensionally preserve' products.

Definition 5. *A cartesian exposure $Q : \mathcal{B} \nrightarrow \mathcal{C}$ of a cartesian P-category $\mathcal{B}$ in a cartesian P-category $\mathcal{C}$ is* product-preserving *whenever the canonical arrows*

$$\langle Q\pi_1, Q\pi_2 \rangle : Q(A \times B) \to QA \times QB$$
$$!_{Q\mathbf{1}} : Q\mathbf{1} \to \mathbf{1}$$

are P-isomorphisms. We write $m_{A,B} : QA \times QB \xrightarrow{\cong} Q(A \times B)$ *and* $m_0 : \mathbf{1} \xrightarrow{\cong} Q\mathbf{1}$ *for their inverses.*

Amongst the exposures, then, the ones that are both cartesian *and* product-preserving are the ones that behave reasonably well in interaction with the product structure. For example, it is an easy calculation to show that

Proposition 1. *In the above setting,* $m_{A,B} \circ \langle Qf, Qg \rangle \sim Q\langle f, g \rangle$.

We can now prove that $\langle f, g \rangle \circ h \approx \langle f \circ h, g \circ h \rangle$:

$$\begin{aligned} Q(\langle f, g \rangle \circ h) &\sim Q(\langle \pi_1 \circ \langle f, g \rangle \circ h, \pi_2 \circ \langle f, g \rangle \circ h \rangle) \\ &\sim m \circ \langle Q(\pi \circ \langle f, g \rangle \circ h), Q(\pi' \circ \langle f, g \rangle \circ h) \rangle \\ &\sim m \circ \langle Q(f \circ h), Q(g \circ h) \rangle \sim Q(\langle f \circ h, g \circ h \rangle) \end{aligned}$$

3.2 Evaluators, Quotation Devices, and Comonadic Exposures

Using transformations of exposures, we may begin to reinterpret concepts from the modality-as-intension interpretation. Throughout this section, we fix a cartesian P-category $\mathfrak{B}$, and a cartesian, product-preserving endoexposure $Q : \mathfrak{B} \nrightarrow \mathfrak{B}$.

Definition 6. *An* evaluator *is a transformation of exposures* $\epsilon : Q \overset{\bullet}{\nrightarrow} \mathsf{Id}_{\mathfrak{B}}$.

What about quoting? Given a point $a : \mathbf{1} \to A$, its *quote* is defined to be the point $Q(a) \circ m_0 : \mathbf{1} \to QA$. We will require the following definition:

Definition 7. *A arrow* $\delta : QA \to Q^2A$ *is a* reasonable quoting device *just if for any* $a : \mathbf{1} \to QA$ *the following diagram commutes up to* $\sim$*:*

$$\begin{array}{ccc} \mathbf{1} & \xrightarrow{a} & QA \\ {\scriptstyle m_0}\downarrow & & \downarrow{\scriptstyle \delta_A} \\ Q\mathbf{1} & \xrightarrow[Qa]{} & Q^2A \end{array}$$

A special case of this condition is the equation that holds if a natural transformation of a similar type to δ is monoidal, namely $\delta_{\mathbf{1}} \circ m_0 \sim Q(m_0) \circ m_0$.

Definition 8. *A* quoter *is a transformation of exposures* $\delta : Q \overset{\bullet}{\nrightarrow} Q^2$ *such that every component* $\delta_A : QA \to Q^2A$ *is a reasonable quoting device.*

These ingredients finally combine to form a *comonadic* exposure.

Definition 9. *A* comonadic exposure (Q, ϵ, δ) *consists of an endoexposure* $Q : (\mathfrak{B}, \sim) \nrightarrow (\mathfrak{B}, \sim)$*, an evaluator* $\epsilon : Q \overset{\bullet}{\nrightarrow} \mathsf{Id}_{\mathfrak{B}}$*, and a quoter* $\delta : Q \overset{\bullet}{\nrightarrow} Q^2$*, such that the following diagrams commute up to* $\sim$*:*

$$\begin{array}{ccc} QA & \xrightarrow{\delta_A} & Q^2A \\ {\scriptstyle \delta_A}\downarrow & & \downarrow{\scriptstyle \delta_{Q(A)}} \\ Q^2A & \xrightarrow[Q(\delta_A)]{} & Q^3A \end{array} \qquad \begin{array}{ccc} QA & \xrightarrow{\delta_A} & Q^2A \\ {\scriptstyle \delta_A}\downarrow & \searrow^{id_A} & \downarrow{\scriptstyle \epsilon_{QA}} \\ Q^2A & \xrightarrow[Q(\epsilon_A)]{} & QA \end{array}$$

4 Exposures and Intensional Recursion

Armed with the above, we can now speak of both extensional and intensional recursion. Lawvere [16] famously proved a theorem which guarantees that, under certain assumptions which we will discuss in Sect. 7, there exist fixed points of the following sort.

Definition 10. *An* extensional fixed point (EFP) *of an arrow* $t : Y \to Y$ *is a point* $y : \mathbf{1} \to Y$ *such that* $t \circ y \sim y$. *If, for a given object* Y, *every arrow* $t : Y \to Y$ *has a EFP, then we say that* Y has EFPs.

In Lawvere's paper EFPs are a kind of fixed point that *oughtn't* exist. In fact, Lawvere shows that—were truth definable—the arrow $\neg : \mathbf{2} \to \mathbf{2}$ representing negation would have a fixed point, i.e. a formula ϕ with $\neg\phi \leftrightarrow \phi$ that leads to inconsistency.

EFPs do not encompass fixed points that *ought* to exist. For example, the *diagonal lemma* for *Peano Arithmetic* (henceforth PA) stipulates that for any predicate $\phi(x)$, there exists a closed formula $\mathbf{fix}(\phi)$ such that

$$\mathsf{PA} \vdash \mathbf{fix}(\phi) \leftrightarrow \phi(\ulcorner \mathbf{fix}(\phi) \urcorner)$$

The formula $\mathbf{fix}(\phi)$ occurs asymmetrically: on the left hand side of the bi-implication it appears as a truth value, but on the right hand side it appears under a *Gödel numbering*, i.e. an assignment $\ulcorner \cdot \urcorner$ of a numeral to each term and formula of PA. Since exposures map values to their encoding, the following notion encompasses this kind of 'asymmetric' fixed point.

Definition 11. *Let* $Q : \mathfrak{B} \nrightarrow \mathfrak{B}$ *be a cartesian, product-preserving endoexposure. An* intensional fixed point (IFP) *of a arrow* $t : QY \to Y$ *is a point* $y : \mathbf{1} \to Y$ *such that*

$$y \sim t \circ Q(y) \circ m_0$$

An object A has IFPs (w.r.t. Q) *if every arrow* $t : QA \to A$ *has a IFP.*

This makes intuitive sense: $y : \mathbf{1} \to Y$ is extensionally equal to t 'evaluated' at the point $Q(y) \circ m_0 : \mathbf{1} \to QY$, which is the 'quoted' version of y.

5 Consistency, Truth and Provability: Gödel and Tarski

We are now in a position to argue that two well-known theorems from logic can be reduced to very simple algebraic arguments involving exposures. In fact, the gist of both arguments relies on the existence of IFPs for an 'object of truth values' in a P-category. The theorems in question are Gödel's *First Incompleteness Theorem* and Tarski's *Undefinability Theorem* [5,24].

Suppose that we have some sort of object $\mathbf{2}$ of 'truth values.' This need not be fancy: we require that it has two points $\top : \mathbf{1} \to \mathbf{2}$ and $\bot : \mathbf{1} \to \mathbf{2}$, standing for true and false respectively. We also require an arrow $\neg : \mathbf{2} \to \mathbf{2}$ which satisfies $\neg \circ \top \sim \bot$ and $\neg \circ \bot \sim \top$.

A simplified version of Gödel's First Incompleteness theorem for PA is this:

Theorem 1 (Gödel). *If PA is consistent, then there are sentences ϕ of PA such that neither $PA \vdash \phi$ nor $PA \vdash \neg\phi$.*

The proof relies on two constructions: the diagonal lemma, and the fact that provability is definable in the system. The definability of provability amounts to the fact that there is a formula $\mathrm{Prov}(x)$ with one free variable x such that $\mathsf{PA} \vdash \phi$ if and only if $\mathsf{PA} \vdash \mathrm{Prov}(\ulcorner\phi\urcorner)$. That is: the system can internally talk about its own provability, modulo some Gödel numbering.

It is not then hard to sketch the proof to Gödel's theorem: take ψ such that $\mathsf{PA} \vdash \psi \leftrightarrow \neg\mathrm{Prov}(\ulcorner\psi\urcorner)$. Then ψ is provable if and only if it is not, so if either $\mathsf{PA} \vdash \psi$ or $\mathsf{PA} \vdash \neg\psi$ we would observe inconsistency. Thus, if PA is consistent, neither ψ nor its negation are provable. It follows that ψ is neither equivalent to $\bot$ or to $\top$. In a way, ψ has an eerie truth value, neither $\top$ nor $\bot$.

Let us represent the provability predicate as an arrow $p : Q\mathbf{2} \to \mathbf{2}$ such that $y \sim \top$ if and only if $p \circ Q(y) \circ m_0 \sim \top$. Consistency is captured by the following definition:

Definition 12. *An object* $\mathbf{2}$ *as above is* simply consistent *just if* $\top \not\sim \bot$.

Armed with this machinery, we can transport the argument underlying Gödel's proof to our more abstract setting:

Theorem 2. *If a $p : Q\mathbf{2} \to \mathbf{2}$ is as above, and $\mathbf{2}$ has IFPs, then one of the following things is true: either (a) there are points of $\mathbf{2}$ other than $\top : \mathbf{1} \to \mathbf{2}$ and $\bot : \mathbf{1} \to \mathbf{2}$; or (b) $\mathbf{2}$ is* not *simply consistent, i.e. $\top \sim \bot$.*

Proof. As $\mathbf{2}$ has IFPs, take $y : \mathbf{1} \to \mathbf{2}$ such that $y \sim \neg \circ p \circ Q(y) \circ m_0$. Now, if $y \sim \top$, then by the property of p above, $p \circ Q(y) \circ m_0 \sim \top$, hence $\neg \circ p \circ Q(y) \circ m_0 \sim \bot$, hence $y \sim \bot$. So either $y \not\sim \top$ or $\mathbf{2}$ is not simply consistent. Similarly, either $y \not\sim \bot$ or $\mathbf{2}$ is not simply consistent.

Tarski's *Undefinability Theorem*, on the other hand is the result that *truth cannot be defined in arithmetic* [24].

Theorem 3 (Tarski). *If PA is consistent, then there is no predicate $True(x)$ such that $PA \vdash \phi \leftrightarrow True(\ulcorner\phi\urcorner)$ for all sentences ϕ.*

The proof is simple: use the diagonal lemma to obtain a closed ψ such that $\mathsf{PA} \vdash \psi \leftrightarrow \neg\mathrm{True}(\ulcorner\psi\urcorner)$, so that $\mathsf{PA} \vdash \psi \leftrightarrow \neg\psi$, which leads to inconsistency.

Now, a proof predicate would constitute an evaluator $\epsilon : Q \overset{\bullet}{\nrightarrow} \mathsf{Id}_{\mathfrak{B}}$: we would have that

$$\epsilon_{\mathbf{2}} \circ Q(y) \circ m_0 \sim y \circ \epsilon_{\mathbf{1}} \circ m_0 \sim y$$

where the last equality is because $\mathbf{1}$ is terminal. This is actually a more general.

Lemma 1. *Let $Q : \mathfrak{B} \nrightarrow \mathfrak{B}$ be an endoexposure, and let $\epsilon : Q \overset{\bullet}{\nrightarrow} Id_{\mathfrak{B}}$ be an evaluator. Then, if A has IFPs then it also has EFPs.*

Proof. Given $t : A \to A$, consider $t \circ \epsilon_A : QA \to A$. A IFP for this arrow is a point $y : \mathbf{1} \to A$ such that $y \sim t \circ \epsilon_A \circ Q(y) \circ m_0$. But we may calculate as above to show that $\epsilon_A \circ Q(y) \circ m_0 \sim y$ and thus $y \sim t \circ y$.

In proving Tarski's theorem, we constructed a sentence ψ such that $\mathsf{PA} \vdash \psi \leftrightarrow \neg\psi$. This can be captured abstractly by the following definition.

Definition 13. *An object* $\mathbf{2}$ *as above is* fix-consistent *just if the arrow* $\neg : \mathbf{2} \to \mathbf{2}$ *has no EFP; that is, there is no* $y : \mathbf{1} \to \mathbf{2}$ *such that* $\neg \circ y \sim y$.

Putting these together, we get

Theorem 4. *If* $\mathbf{2}$ *has IFPs in the presence of an evaluator, then it is* not *fix-consistent.*

6 An Exposure on Arithmetic

We will substantiate the results of the previous section by sketching the construction of a P-category and endoexposure based on a first-order theory. The method is very similar to that of Lawvere [16], and we will also call it the *Lindenbaum P-category* of the theory. The construction is general, and so is the thesis of this section: an exposure on a Lindenbaum P-category abstractly captures the notion of a well-behaved Gödel 'numbering' on the underlying theory.

Let there be a single-sorted first-order theory T. The objects of the P-category are the formal products of (a) $\mathbf{1}$, the terminal object, (b) A, the *domain*, and (c) $\mathbf{2}$, the object of *truth values*. Arrows $\mathbf{1} \to A$ and $A \to A$ are *terms* with no or one free variable respectively. Arrows $A^n \to \mathbf{2}$ and $\mathbf{1} \to \mathbf{2}$ are *predicates*, with n and no free variables respectively. Finally, arrows $\mathbf{2}^n \to \mathbf{2}$ can be thought of as *logical connectives* (e.g. $\wedge : \mathbf{2} \times \mathbf{2} \to \mathbf{2}$).

Two arrows $s, t : C \to A$ with codomain A (i.e. two terms of the theory) are related if and only if they are provably equal, i.e. $s \sim t$ iff $\mathsf{T} \vdash s = t$. Two arrows $\phi, \psi : C \to \mathbf{2}$ with codomain $\mathbf{2}$ are related if and only if they are provably equivalent, i.e. $\phi \sim \psi$ iff $\mathsf{T} \vdash \phi \leftrightarrow \psi$.

To define an exposure, it suffices to have a Gödel numbering, i.e. a representation of terms and formulas of the theory as elements of its domain A. More precisely, we need a Gödel numbering for which *substitution is internally definable*. We write $\ulcorner \phi(x_1, \ldots, x_n) \urcorner$ and $\ulcorner t(a_1, \ldots, a_m) \urcorner$ for the Gödel numbers of the formula $\phi(x_1, \ldots, x_n)$ and the term $t(a_1, \ldots, a_m)$ respectively, and we assume that $\ulcorner \cdot \urcorner$ is injective. Let $QA \stackrel{\text{def}}{=} A$, $Q(\mathbf{2}) \stackrel{\text{def}}{=} A$, and $Q(\mathbf{1}) \stackrel{\text{def}}{=} \mathbf{1}$. Finally, define Q to act component-wise on finite products: this will guarantee that it is cartesian and product-preserving.

The action on arrows is what necessitated that substitution be definable: this amounts to the existence of a term $sub(y, x)$ with the property that if $\phi(x)$ is a predicate and t is a term, then $\mathsf{T} \vdash sub(\ulcorner \phi \urcorner, \ulcorner t \urcorner) = \ulcorner \phi(t) \urcorner$. Now, given a predicate $\phi : A \to \mathbf{2}$ with one free variable, $Q(\phi) : A \to A$ is defined to be the term $sub(\ulcorner \phi \urcorner, x)$. Given a sentence $\phi : \mathbf{1} \to \mathbf{2}$, we define $Q(\phi) : \mathbf{1} \to A$ to be

exactly the closed term $\ulcorner\phi\urcorner$. The action is similar on arrows with codomain A, and component-wise on product arrows. The last axiom of exposures is satisfied: if $Q\phi \sim Q\psi$, then $\ulcorner\phi\urcorner = \ulcorner\psi\urcorner$, so that $\phi = \psi$, by the injectivity of the Gödel numbering.

In this setting, IFPs really are fixpoints of formulas.

7 Where Do IFPs Come From?

In Sect. 4 we mentioned Lawvere's fixed point theorem. This theorem guarantees the existence of EFPs under the assumption that there is an arrow of this form:

Definition 14. *An arrow $r : X \times A \to Y$ is* weakly-point surjective *if, for every $f : A \to Y$, there exists a $x_f : \mathbf{1} \to X$ such that for all points $a : \mathbf{1} \to A$ it is the case that $r \circ \langle x_f, a\rangle \sim f \circ a$.*

So a weak-point surjection is a bit like 'pointwise cartesian closure,' in that the effect of all arrows $A \to Y$ on points $\mathbf{1} \to A$ is representable by some point $\mathbf{1} \to X$, w.r.t. r. Lawvere noticed that if the 'exponential' X and the domain A coincide, then a simple diagonal argument yields fixpoints for all arrows $Y \to Y$.

Theorem 5 (Lawvere). *If $r : A \times A \to Y$ is a weak-point surjection, then every arrow $t : Y \to Y$ has an extensional fixed point (EFP).*

Proof. Let $f \stackrel{\text{def}}{=} t \circ r \circ \langle id_A, id_A\rangle$. Then there exists a $x_f : \mathbf{1} \to A$ such that $r \circ \langle x_f, a\rangle \sim f \circ a$ for all $a : \mathbf{1} \to A$. We compute that $r \circ \langle x_f, x_f\rangle \sim\ t \circ r \circ \langle id_A, id_A\rangle \circ x_f \sim\ t \circ r \circ \langle x_f, x_f\rangle$, so that $r \circ \langle x_f, x_f\rangle$ is a EFP of t.

Can we adapt Lawvere's result to IFPs? The answer is positive, and rather straightforward once we embellish the statement with appropriate occurrences of Q. We also need a reasonable quoting device.

Theorem 6. *Let Q be a monoidal exposure, and let $\delta_A : QA \to Q^2A$ be a reasonable quoting device. If $r : QA \times QA \to Y$ is a weak-point surjection then every arrow $t : QY \to Y$ has an intensional fixed point.*

Proof. Let $f \stackrel{\text{def}}{=} t \circ Qr \circ m_{QA,QA} \circ \langle \delta_A, \delta_A\rangle$. Then there exists a $x_f : \mathbf{1} \to QA$ such that $r \circ \langle x_f, a\rangle \sim f \circ a$ for all $a : \mathbf{1} \to QA$. We compute that

$$\begin{aligned} r \circ \langle x_f, x_f\rangle &\sim t \circ Qr \circ m \circ \langle \delta_A, \delta_A\rangle \circ x_f \sim\ t \circ Qr \circ m \circ \langle \delta_A \circ x_f, \delta_A \circ x_f\rangle \\ &\sim t \circ Qr \circ m \circ \langle Q(x_f) \circ m_0, Q(x_f) \circ m_0\rangle \\ &\sim t \circ Qr \circ m \circ \langle Q(x_f), Q(x_f)\rangle \circ m_0 \\ &\sim t \circ Qr \circ Q(\langle x_f, x_f\rangle) \circ m_0 \sim\ t \circ Q(r \circ \langle x_f, x_f\rangle) \circ m_0 \end{aligned}$$

so that $r \circ \langle x_f, x_f\rangle$ is a IFP of t.

In the next section, we shall see that this is a true categorical analogue of Kleene's *Second Recursion Theorem (SRT)*.

8 The Recursion Theorems

In fact, the theorem we just proved in Sect. 7 is strongly reminiscent of a known theorem in (higher order) computability theory, namely a version of Kleene's *First Recursion Theorem (FRT)*.

Let us fix some notation. We write $\simeq$ for *Kleene equality*: we write $e \simeq e'$ to mean either that both expressions e and e' are undefined, if either both are undefined, or both are defined and of equal value. Let ϕ_0, ϕ_1, ... be an enumeration of the partial recursive functions. We will also require the *s-m-n theorem* from computability theory. Full definitions and statements may be found in the book by Cutland [9].

Theorem 7 (First Recursion Theorem). *Let $\mathcal{PR}$ be the set of unary partial recursive functions, and let $F : \mathcal{PR} \to \mathcal{PR}$ be an* effective operation. *Then $F : \mathcal{PR} \to \mathcal{PR}$ has a fixed point.*

Proof. That $F : \mathcal{PR} \to \mathcal{PR}$ is an effective operation means that there is a partial recursive $f : \mathbb{N} \times \mathbb{N} \rightharpoonup \mathbb{N}$ such that $f(e, x) \simeq F(\phi_e)(x)$. Let $d \in \mathbb{N}$ a code for the partial recursive function $\phi_d(y, x) \stackrel{\text{def}}{=} f(S(y, y), x)$, where $S : \mathbb{N} \times \mathbb{N} \rightharpoonup \mathbb{N}$ is the s-1-1 function of the s-m-n theorem. Then, by the s-m-n theorem, and the definitions of $d \in \mathbb{N}$ and f,

$$\phi_{S(d,d)}(x) \simeq \phi_d(d, x) \simeq f(S(d, d), x) \simeq F(\phi_{S(d,d)})(x)$$

so that $\phi_{S(d,d)}$ is a fixed point of $F : \mathcal{PR} \to \mathcal{PR}$.

Lawvere's theorem is virtually identical to a point-free version of this proof. Yet, one cannot avoid noticing that we have proved more than that for which we bargained. The $f : \mathbb{N} \times \mathbb{N} \rightharpoonup \mathbb{N}$ in the proof above has the special property that it is *extensional*, in the sense that

$$\phi_e = \phi_{e'} \quad \Longrightarrow \quad \forall x \in \mathbb{N}.\ f(e, x) = \phi(e', x)$$

However, the step which yields the fixed point argument holds for any such f, not just the extensional ones. This fact predates the FRT, and was shown by Kleene in 1938 [14].

Theorem 8 (Second Recursion Theorem). *For any partial recursive $f : \mathbb{N} \times \mathbb{N} \rightharpoonup \mathbb{N}$, there exists $e \in \mathbb{N}$ such that $\phi_e(y) \simeq f(e, y)$ for all $y \in \mathbb{N}$.*

This is significantly more powerful than the FRT, as $f(e, y)$ can make arbitrary decisions depending on the source code e, irrespective of the function ϕ_e of which it is the source code. Moreover, it is evident that the function ϕ_e has *access to its own code*, allowing for a certain degree of reflection. Even if f is extensional, hence defining an effective operation, the SRT grants us more power than the FRT: for example, before recursively calling e on some points, $f(e, y)$ could 'optimise' e depending on what y is, hence ensuring that the recursive call

will run faster than e itself would. This line of thought is common in the *partial evaluation* community, see e.g. [12].

In the sequel we argue that our fixed point theorem involving exposures is a generalisation of Lawvere's theorem, in the exact same way that the SRT is a non-extensional generalisation of the FRT. In order to do so, we define a P-category and an exposure based on realizability theory, and claim that the FRT and the SRT are instances of the general theorems in that particular P-category.

9 An Exposure on Assemblies

Our second example of an exposure will come from realizability, where the basic objects are assemblies. An assembly is a set to every element of which we have associated a set of *realizers*. The elements of the set can be understood as *elements of a datatype*, and the set of realizers of each such element as the *machine-level representations* of it. For example, if realizers range in the natural numbers, then assemblies and functions between them which are partial recursive on the level of realizers yield a category where 'everything is computable.'

In practice, the generalisation from natural numbers to an arbitrary *partial combinatory algebra (PCA)* is made. A PCA is an arbitrary, untyped 'universe' corresponding to some notion of computability or realizability. There are easy tricks with which one may encode various common 'first-order' datatypes, such as booleans, integers, etc. as well as all partial recursive functions (up to the encoding of integers). These methods can be found [3,17,20,21].

Definition 15. *A* partial combinatory algebra (PCA) $(A, \cdot)$ *consists of a set* A*, its carrier, and a partial binary operation* $\cdot : A \times A \rightharpoonup A$ *such that there exist* $\mathbf{K}, \mathbf{S} \in A$ *with the properties that*

$$\mathbf{K} \cdot x \downarrow, \quad \mathbf{K} \cdot x \cdot y \simeq y, \quad \mathbf{S} \cdot x \cdot y \downarrow, \quad \mathbf{S} \cdot x \cdot y \cdot z \simeq x \cdot z \cdot (y \cdot z)$$

for all $x, y, z \in A$*.*

The simplest example of a PCA, corresponding to classical computability, is K_1, also known as *Kleene's first model*. Its carrier is $\mathbb{N}$, and $r \cdot a \stackrel{\text{def}}{=} \phi_r(a)$.

Definition 16. *An* assembly X *on a PCA* A *consists of a set* $|X|$ *and, for each* $x \in |X|$*, a non-empty subset* $\|x\|_X$ *of* A*. If* $a \in \|x\|_X$*, we say that* a realizes x*.*

Definition 17. *For two assemblies* X *and* Y*, a function* $f : |X| \to |Y|$ *is said to be* tracked by $r \in A$ *just if, for all* $x \in |X|$ *and* $a \in \|x\|_X$*, we have* $r \cdot a \downarrow$ *and* $r \cdot a \in \|f(x)\|_Y$*.*

Now: for each PCA A, we can define a category $\mathbf{Asm}(A)$, with objects all assemblies X on A, and morphisms $f : X \to Y$ all functions $f : |X| \to |Y|$ that are tracked by some $r \in A$.

Theorem 9. *Assemblies and 'trackable' morphisms between them form a category* $\mathbf{Asm}(A)$ *that is cartesian closed, has finite coproducts, and a natural numbers object.*

We only mention one other construction that we shall need. Given an assembly X, the *lifted assembly* $X_\perp$ is defined to be

$$|X_\perp| \stackrel{\text{def}}{=} |X| \cup \{\perp\} \quad \text{and} \quad \|x\|_{X_\perp} \stackrel{\text{def}}{=} \begin{cases} \{\, r \mid r \cdot \overline{0} \downarrow \text{ and } r \cdot \overline{0} \in \|x\|_X \,\} & \text{for } x \in |X| \\ \{\, r \mid r \cdot \overline{0} \uparrow \,\} & \text{for } x = \perp \end{cases}$$

for some chosen element of the PCA $\overline{0}$. Elements of $X_\perp$ are either elements of X, or the undefined value $\perp$. Realizers of $x \in |X|$ are 'computations' $r \in A$ which, when run (i.e. given the dummy value $\overline{0}$ as argument) return a realizer of x. A computation that does not halt when run represents the undefined value.[2]

9.1 Passing to a P-category

The lack of intensionality in the category $\mathbf{Asm}(A)$ is blatantly obvious. To elevate a function $f : |X| \to |Y|$ to a morphism $f : X \to Y$, we only require that *there exists* a 'witness' $r \in A$ that realizes it, and then we forget about this witness entirely. To mend this, we define a P-category.

The P-category $\mathfrak{Asm}(A)$ of assemblies on A is defined to have all assemblies X on A as objects, and pairs $(f : |X| \to |Y|, r \in A)$ where r tracks f as arrows. We define $(f, r) \sim (g, s)$ just if $f = g$, i.e. when the underlying function is the same. The composition of $(f, p) : X \to Y$ and $(g, q) : Y \to Z$ is $(g \circ f, \mathbf{B} \cdot q \cdot p)$ where $\mathbf{B}$ is a combinator in the PCA such that $\mathbf{B} \cdot f \cdot g \cdot x \simeq f \cdot (g \cdot x)$ for any $f, g, x \in A$. The *identity* $id_X : X \to X$ is defined to be $(id_{|X|}, \mathbf{I}) : X \to X$, where $\mathbf{I}$ is a combinator in the PCA such that $\mathbf{I} \cdot x \simeq x$ for all $x \in A$.

Much in the same way as before—but now up to the PER $\sim$—we can show

Theorem 10. $\mathfrak{Asm}(A)$ *is a cartesian closed P-category with a natural numbers object* $\mathbb{N}$.

We can now define an exposure $\Box : \mathfrak{Asm}(A) \nrightarrow \mathfrak{Asm}(A)$. For an assembly $X \in \mathfrak{Asm}(A)$, let $\Box X$ be the assembly defined by

$$|\Box X| \stackrel{\text{def}}{=} \{\, (x, a) \mid x \in |X|, a \in \|x\|_A \,\}, \quad \|(x, a)\|_{\Box X} \stackrel{\text{def}}{=} \{\, a \,\}$$

Given $(f, r) : X \to Y$, we define $\Box(f, r) = (f_r, r) : \Box X \to \Box Y$ where $f_r : |\Box X| \to |\Box Y|$ is defined by $f_r(x, a) \stackrel{\text{def}}{=} (f(x), r \cdot a)$. Thus, under the exposure each element $(x, a) \in |\Box X|$ carries with it its own unique realizer a. The image of (f, r) under $\Box$ shows not only what f does to an element of its domain, but also how r acts on the realizer of that element.

It is long but straightforward to check that

[2] Bear in mind that this definition of the lifted assembly does not work if the PCA is total. We are mostly interested in the decidedly non-total PCA K_1, so this is not an issue. There are other, more involved ways of defining the lifted assembly; see [20] in particular.

Theorem 11. $\Box : \mathfrak{Asm}(A) \nrightarrow \mathfrak{Asm}(A)$ *is a cartesian, product-preserving, and comonadic endoexposure.*

9.2 Kleene's Recursion Theorems, Categorically

Let us concentrate on the category $\mathfrak{Asm}(K_1)$. Arrows $\mathbb{N} \to \mathbb{N}_\perp$ are easily seen to correspond to partial recursive functions. It is not hard to produce a weak-point surjection $r_E : \mathbb{N} \times \mathbb{N} \to \mathbb{N}_\perp^{\mathbb{N}}$, and hence to invoke Lawvere's theorem to show that every arrow $\mathbb{N}_\perp^{\mathbb{N}} \to \mathbb{N}_\perp^{\mathbb{N}}$ has an extensional fixed point. Now, by Longley's generalised Myhill-Shepherdson theorem [17,19], arrows $\mathbb{N}_\perp^{\mathbb{N}} \to \mathbb{N}_\perp^{\mathbb{N}}$ correspond to effective operations. Hence, in this context Lawvere's theorem corresponds to the simple diagonal argument that we used to show the FRT.[3]

Let us look at arrows of type $\Box(\mathbb{N}_\perp^{\mathbb{N}}) \to \mathbb{N}_\perp^{\mathbb{N}}$. These correspond to 'non-functional' transformations, mapping functions to functions, but without respecting extensionality. As every natural number indexes a partial recursive function, these arrows really correspond to all partial recursive functions (up to some tagging and encoding). It is not hard to see that $\Box\mathbb{N}$ is P-isomorphic to $\mathbb{N}$, and that one can build a weak-point surjection of type $\Box\mathbb{N} \times \Box\mathbb{N} \to \mathbb{N}_\perp^{\mathbb{N}}$, so that by our theorem, every arrow of type $\Box(\mathbb{N}_\perp^{\mathbb{N}}) \to \mathbb{N}_\perp^{\mathbb{N}}$ has an intensional fixed point. This is exactly Kleene's SRT!

10 Rice's Theorem

To further illustrate the applicability of the language of exposure, we state and prove an abstract version of *Rice's theorem*. Rice's theorem is a result in computability which states that no computer can decide any non-trivial property of a program by looking at its code. A short proof relies on the SRT.

Theorem 12 (Rice). *Let $\mathcal{F}$ be a non-trivial set of partial recursive functions, and let $A_{\mathcal{F}} \stackrel{def}{=} \{\, e \in \mathbb{N} \mid \phi_e \in \mathcal{F} \,\}$ be the set of indices of functions in that set. Then $A_{\mathcal{F}}$ is undecidable.*

Proof. Suppose $A_{\mathcal{F}}$ is decidable. The fact $\mathcal{F}$ is non-trivial means that there is some $a \in \mathbb{N}$ such that $\phi_a \in \mathcal{F}$ and some $b \in \mathbb{N}$ such that $\phi_b \notin \mathcal{F}$. Consequently, $a \in A_{\mathcal{F}}$ and $b \notin A_{\mathcal{F}}$.

Define $f(e, x) \simeq \textbf{if}\, e \in A_{\mathcal{F}}\, \textbf{then}\, \phi_b(x)\, \textbf{else}\, \phi_a(x)$. By Church's thesis, $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is partial recursive. Use the SRT to obtain $e \in \mathbb{N}$ such that $\phi_e(x) \simeq f(e, x)$. Now, either $e \in A_{\mathcal{F}}$ or not. If it is, $\phi_e(x) \simeq f(e, x) \simeq \phi_b(x)$, so that $\phi_e \notin \mathcal{F}$, a contradiction. A similar phenomenon occurs if $e \notin A_{\mathcal{F}}$.

Constructing the function f in the proof required three basic elements: (a) the ability to evaluate either ϕ_a or ϕ_b given a and b; (b) the ability to

[3] But note that this is not the complete story, as there is no guarantee that the fixed point obtained in *least*, which is what Kleene's original proof in [15] gives. See also [13].

decide which one to use depending on the input; and (c) intensional recursion. For (a), we shall need evaluators, for (b) we shall need that the truth object $\mathbf{2}$ is a *weak coproduct* of two copies of $\mathbf{1}$, and for (c) we shall require IFPs.

Theorem 13. *Let* $\mathbf{2}$ *is a simply consistent 'truth object' which also happens to be a weak coproduct of two copies of* $\mathbf{1}$*, with injections* $\top : \mathbf{1} \to \mathbf{2}$ *and* $\bot : \mathbf{1} \to \mathbf{2}$*. Furthermore, suppose that* A *has EFPs. If* $f : A \to \mathbf{2}$ *is such that for all* $x : \mathbf{1} \to A$*, either* $f \circ x \sim \top$ *or* $f \circ x \sim \bot$*, then* f *is trivial, in the sense that either* $f \circ x \sim \top$ *for all* $x : \mathbf{1} \to A$*, or* $f \circ x \sim \bot$ *for all* $x : \mathbf{1} \to A$*.*

Proof. Suppose there are two such distinct $a, b : \mathbf{1} \to A$ such that $f \circ a \sim \top$ and $f \circ b \sim \bot$. Let $g \stackrel{\text{def}}{=} [b, a] \circ f$ and let $y : \mathbf{1} \to A$ be its EFP. Now, either $f \circ y \sim \top$ or $f \circ y \sim \bot$. In the first case, we can calculate that $\top \sim \ f \circ [b, a] \circ f \circ y \sim f \circ [b, a] \circ \top \sim \ f \circ b \sim \ \bot$ so that $\mathbf{2}$ is *not simply consistent.* A similar situation occurs if $f \circ y \sim \bot$.

Needless to say that the premises of this theorem are easily satisfied in our exposure on assemblies from Sect. 9 if we take $A = {\mathbb{N}_\bot}^{\mathbb{N}}$ and $\mathbf{2}$ to be the lifted coproduct $(\mathbf{1} + \mathbf{1})_\bot$.

11 Conclusion

We have modelled intensionality with P-categories, and introduced a new construct that abstractly corresponds to Gödel numbers. This led us to an immediate unification of many 'diagonal arguments' in logic and computability, as well as a new perspective on the notion of *intensional recursion.* Our approach is clearer and more systematic than the one in [16].

Many questions are left open. We are currently working on the medium-term goal of a safe, reflective programming language based on modal type theory. The basics are there, but there are many questions: what operations should be available at modal types; with how much expressivity would the language be endowed for each possible set; and what are the applications?

On the more technical side, it is interesting to note that we have refrained from a categorical proof of the diagonal lemma for PA. All our attempts were inelegant, and we believe that this is because arithmetic is fundamentally untyped: $Q(\mathbf{2})$ has many more points than 'all Gödel numbers of predicates.' In contrast, our approach using exposures is typed, which sets it apart from all previous attempts at capturing such arguments categorically, including the very elegant work of Cockett and Hofstra [6,7]. The approach in *op. cit.* is based on *Turing categories*, in which every object is a retract of some very special objects—the *Turing objects.* In the conclusion of [6] this is explicitly mentioned as an 'inherent limitation.' Only time will tell which approach is more encompassing.

Finally, it would be interesting to study the meaning of exposure in examples not originating in logic and computability, but in other parts of mathematics. Can we find examples of exposures elsewhere? Are they of any use?

Acknowledgements. I would like to thank my doctoral supervisor, Samson Abramsky, for suggesting the topic of this paper, and for his help in understanding the issues around intensionality and intensional recursion.

References

1. Abramsky, S.: Intensionality, definability and computation. In: Baltag, A., Smets, S. (eds.) Johan van Benthem on Logic and Information Dynamics. OCL, vol. 5, pp. 121–142. Springer, Cham (2014). doi:10.1007/978-3-319-06025-5_5
2. Barendregt, H.: Self-interpretation in lambda calculus. J. Funct. Program. **1**(2), 229–233 (1991). https://dx.doi.org/10.1017/S0956796800020062
3. Beeson, M.J.: Foundations of Constructive Mathematics. Springer, Heidelberg (1985). https://dx.doi.org/10.1007/978-3-642-68952-9
4. Bierman, G.M., de Paiva, V.: On an intuitionistic modal logic. Stud. Logica **65**(3), 383–416 (2000). https://dx.doi.org/10.1023/A:1005291931660
5. Boolos, G.S.: The Logic of Provability. Cambridge University Press, Cambridge (1994). https://dx.doi.org/10.1017/CBO9780511625183
6. Cockett, J.R.B., Hofstra, P.J.W.: Introduction to Turing categories. Ann. Pure Appl. Logic **156**(2–3), 183–209 (2008). http://dx.doi.org/10.1016/j.apal.2008.04.005
7. Cockett, J.R.B., Hofstra, P.J.W.: Categorical simulations. J. Pure Appl. Algebra **214**(10), 1835–1853 (2010). http://dx.doi.org/10.1016/j.jpaa.2009.12.028
8. Čubrić, D., Dybjer, P., Scott, P.J.: Normalization and the Yoneda embedding. Math. Struct. Comput. Sci. **8**(2), 153–192 (1998). https://dx.doi.org/10.1017/s0960129597002508
9. Cutland, N.: Computability: An Introduction to Recursive Function Theory. Cambridge University Press, Cambridge (1980)
10. Davies, R., Pfenning, F.: A modal analysis of staged computation. J. ACM **48**(3), 555–604 (2001). http://dl.acm.org/citation.cfm?id=382785
11. Fitting, M.: Intensional logic. In: Zalta, E.N. (ed.) The Stanford Encyclopedia of Philosophy. Metaphysics Research Lab, Stanford University, Summer 2015 edn. (2015). https://plato.stanford.edu/archives/sum2015/entries/logic-intensional/
12. Jones, N.D.: An introduction to partial evaluation. ACM Comput. Surv. **28**(3), 480–503 (1996). http://doi.acm.org/10.1145/243439.243447
13. Kavvos, G.A.: Kleene's two kinds of recursion. CoRR abs/1602.06220 (2016). http://arxiv.org/abs/1602.06220
14. Kleene, S.C.: On notation for ordinal numbers. J. Symbolic Logic **3**(04), 150–155 (1938). https://dx.doi.org/10.2307/2267778
15. Kleene, S.C.: Introduction to Metamathematics. North-Holland, Amsterdam (1952)
16. Lawvere, F.W.: Diagonal arguments and cartesian closed categories. Reprints in Theory and Applications of Categories **15**, 1–13 (2006). http://www.tac.mta.ca/tac/reprints/articles/15/tr15abs.html
17. Longley, J.R.: Realizability toposes and language semantics. Ph.D. thesis, University of Edinburgh. College of Science and Engineering. School of Informatics (1995). http://www.lfcs.inf.ed.ac.uk/reports/95/ECS-LFCS-95-332/
18. Longley, J.R.: Notions of computability at higher types I. In: Logic Colloquium 2000: Proceedings of the Annual European Summer Meeting of the Association for Symbolic Logic. Lecture Notes in Logic, 23–31 July Paris, France, vol. 19, pp. 32–142. A. K. Peters (2005)

19. Longley, J.R., Normann, D.: Higher-Order Computability. Theory and Applications of Computability. Springer, Heidelberg (2015). https://dx.doi.org/10.1007/978-3-662-47992-6
20. Longley, J.R., Simpson, A.K.: A uniform approach to domain theory in realizability models. Math. Struct. Comput. Sci. **7**, 469–505 (1997). https://dx.doi.org/10.1017/S0960129597002387
21. van Oosten, J.: Realizability: An Introduction to its Categorical Side, vol. 152. Elsevier (2008). http://www.sciencedirect.com/science/bookseries/0049237X/152
22. Pfenning, F., Davies, R.: A judgmental reconstruction of modal logic. Math. Struct. Comput. Sci. **11**(4), 511–540 (2001). https://dx.doi.org/10.1017/S0960129501003322
23. Smith, B.C.: Reflection and semantics in LISP. In: Proceedings of the 11th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL 1984), pp. 23–35. ACM Press, New York (1984). https://dx.doi.org/10.1145/800017.800513
24. Smullyan, R.M.: Gödel's Incompleteness Theorems. Oxford University Press, New York (1992)
25. Wand, M.: The theory of fexprs is trivial. LISP Symbolic Comput. **10**(3), 189–199 (1998). https://dx.doi.org/10.1023/A:1007720632734

Author Index