

8th Conference on Algebra and Coalgebra in Computer Science

CALCO 2019, June 3–6, 2019, London, United Kingdom

Edited by

Markus Roggenbach

Ana Sokolova



Editors

Markus Roggenbach

Swansea University, UK
m.roggenbach@swansea.ac.uk

Ana Sokolova

University of Salzburg, Austria
ana.sokolova@cs.uni-salzburg.at

ACM Classification 2012

Theory of computation → Models of computation; Theory of computation → Modal and temporal logics;
Theory of computation → Algebraic semantics; Theory of computation → Categorical semantics; Theory
of computation → Quantum computation theory; Software and its engineering → Specification languages

ISBN 978-3-95977-120-7

Published online and open access by

Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern,
Germany. Online available at <https://www.dagstuhl.de/dagpub/978-3-95977-120-7>.

Publication date

November, 2019

Bibliographic information published by the Deutsche Nationalbibliothek

The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed
bibliographic data are available in the Internet at <https://portal.dnb.de>.

License

This work is licensed under a Creative Commons Attribution 3.0 Unported license (CC-BY 3.0):
<https://creativecommons.org/licenses/by/3.0/legalcode>.



In brief, this license authorizes each and everybody to share (to copy, distribute and transmit) the work
under the following conditions, without impairing or restricting the authors' moral rights:

- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/LIPIcs.CALCO.2019.0

ISBN 978-3-95977-120-7

ISSN 1868-8969

<https://www.dagstuhl.de/lipics>

LIPICs – Leibniz International Proceedings in Informatics

LIPICs is a series of high-quality conference proceedings across all fields in informatics. LIPICs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

Editorial Board

- Luca Aceto (*Chair*, Gran Sasso Science Institute and Reykjavik University)
- Christel Baier (TU Dresden)
- Mikolaj Bojanczyk (University of Warsaw)
- Roberto Di Cosmo (INRIA and University Paris Diderot)
- Javier Esparza (TU München)
- Meena Mahajan (Institute of Mathematical Sciences)
- Dieter van Melkebeek (University of Wisconsin-Madison)
- Anca Muscholl (University Bordeaux)
- Luke Ong (University of Oxford)
- Catuscia Palamidessi (INRIA)
- Thomas Schwentick (TU Dortmund)
- Raimund Seidel (Saarland University and Schloss Dagstuhl – Leibniz-Zentrum für Informatik)

ISSN 1868-8969

<https://www.dagstuhl.de/lipics>

■ Contents

| | |
|---|----------|
| Preface | |
| <i>Markus Roggenbach and Ana Sokolova</i> | 0:vii |
| Conference Organization | |
| | 0:ix–0:x |

Invited Papers

| | |
|--|---------|
| Matching μ -Logic: Foundation of \mathbb{K} Framework | |
| <i>Xiaohong Chen and Grigore Roşu</i> | 1:1–1:4 |
| From Equational Specifications of Algebras with Structure to Varieties of Data Languages | |
| <i>Stefan Milius</i> | 2:1–2:5 |
| Principles of Natural Language, Logic, and Tensor Semantics | |
| <i>Mehrnoosh Sadrzadeh</i> | 3:1–3:4 |
| Coinduction: Automata, Formal Proof, Companions | |
| <i>Damien Pous</i> | 4:1–4:4 |

Regular Papers

| | |
|---|------------|
| Ω -Automata: A Coalgebraic Perspective on Regular ω -Languages | |
| <i>Vincenzo Ciancia and Yde Venema</i> | 5:1–5:18 |
| Tree Automata as Algebras: Minimisation and Determinisation | |
| <i>Gerco van Heerdt, Tobias Kappé, Jurriaan Rot, Matteo Sammartino, and Alexandra Silva</i> | 6:1–6:22 |
| Coalgebraic Geometric Logic | |
| <i>Nick Bezhanishvili, Jim de Groot, and Yde Venema</i> | 7:1–7:18 |
| Coinduction in Flow: The Later Modality in Fibrations | |
| <i>Henning Basold</i> | 8:1–8:22 |
| Causal Unfoldings | |
| <i>Marc de Visme and Glynn Winskel</i> | 9:1–9:18 |
| A Coalgebraic Perspective on Probabilistic Logic Programming | |
| <i>Tao Gu and Fabio Zanasi</i> | 10:1–10:21 |
| Sequencing and Intermediate Acceptance: Axiomatisation and Decidability of Bisimilarity | |
| <i>Astrid Belder, Bas Luttik, and Jos Baeten</i> | 11:1–11:22 |
| On Terminal Coalgebras Derived from Initial Algebras | |
| <i>Jiří Adámek</i> | 12:1–12:21 |
| Coinductive Resumption Monads: Guarded Iterative and Guarded Elgot | |
| <i>Paul Blain Levy and Sergey Goncharov</i> | 13:1–13:17 |

8th Conference on Algebra and Coalgebra in Computer Science (CALCO 2019).

Editors: Markus Roggenbach and Ana Sokolova



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

| | |
|---|------------|
| Decomposing Comonad Morphisms <i>Danel Ahman and Tarmo Uustalu</i> | 14:1–14:19 |
| The Axiom of Choice in Cartesian Bicategories <i>Filippo Bonchi, Jens Seeber, and Pawel Sobociński</i> | 15:1–15:17 |
| Linear-Time Graph Algorithms in GP 2 <i>Graham Campbell, Brian Courtehoue, and Detlef Plump</i> | 16:1–16:23 |

Tool Paper

| | |
|--|------------|
| Hybridisation of Institutions in HETS <i>Mihai Codrescu</i> | 17:1–17:10 |
|--|------------|

Regular Papers

| | |
|---|------------|
| Nominal String Diagrams <i>Samuel Balco and Alexander Kurz</i> | 18:1–18:20 |
| A Diagrammatic Approach to Quantum Dynamics <i>Stefano Gogioso</i> | 19:1–19:23 |

Tool Paper

| | |
|---|-----------|
| CARTOGRAPHER: A Tool for String Diagrammatic Reasoning <i>Pawel Sobociński, Paul W. Wilson, and Fabio Zanasi</i> | 20:1–20:7 |
|---|-----------|

■ Preface

This volume contains the proceedings of the 8th Conference on Algebra and Coalgebra in Computer Science (CALCO), held at University College London, London, UK, from June 3 to June 6, 2019. CALCO took place under the auspices of IFIP WG 1.3 “Foundations of System Specification”. Local organizers were Philipa Gardner, Emanuele D’Osualdo, Alexandra Silva, and Fabio Zanasi. Henning Basold was publicity chair. CALCO was co-located with the conference Mathematical Foundations of Programming Semantics (MFPS).

At the conference were four invited talks, by Stefan Milius, Damien Pous, Grigore Rosu, and Mehrnoosh Sadrzadeh. Damien Pous and Mehrnoosh Sadrzadeh were joint invited speakers for CALCO and MFPS, and the invited talk by Damien Pous was accompanied by a special session on Coinduction organized by Damien with invited talks by Chung-Kil Hur and Andrei Popescu. There were further 20 contributed talks, of which 14 were full papers, 2 tool papers, and 6 early ideas. Areas covered included Automata, Logics, Causality, Behaviour, Categories, Graphs, and Strings. This volume collects abstracts for the four invited talks, the peer reviewed full papers, and the peer reviewed tool papers. Further details on CALCO are featured on the conference website (<https://www.coalg.org/calco-mfps-2019/calco/>).

CALCO 2019 received submissions from Argentina, Australia, Czechia, Egypt, France, Germany, Iceland, Italy, Japan, the Netherlands, Norway, Slovenia, Switzerland, the United Kingdom, and the United States. In total, there were 30 submissions, of which 21 were full papers, 2 were tool papers and 7 were early ideas submissions.

CALCO is a high-level, bi-annual conference formed by joining the forces and reputations of CMCS (the International Workshop on Coalgebraic Methods in Computer Science), and WADT (the Workshop on Algebraic Development Techniques). It provides a forum to present and discuss results of theoretical nature on the mathematics of algebras and coalgebras, the way these results can support methods and techniques for software development, as well as experience reports concerning the transfer of the resulting technologies into industrial practice. Typical topics of interest include:

- Abstract models and logics
- Algebraic and coalgebraic semantics
- Corecursion in programming languages
- Algebraic and coalgebraic methods in software and systems engineering
- Specialised models and calculi
- String diagrams and network theory
- System specification and verification
- Tools supporting algebraic and coalgebraic methods
- Quantum computing with algebra and coalgebra

CALCO can look back on a proud history: previous CALCO editions took place in Swansea (Wales, 2005), Bergen (Norway, 2007), Udine (Italy, 2009), Winchester (UK, 2011), Warsaw (Poland, 2013), Nijmegen (The Netherlands, 2015) and Ljubljana (Slovenia, 2017).

This volume first presents abstracts of the four invited talks, followed by the contributed papers and tool papers in the order they were presented at the conference. We hope that reading the contributions in this volume will be as inspirational as listening to the talks was in June 2019 in London.

October 2019

Ana Sokolova
Markus Roggenbach



■ Conference Organization

Steering Committee

Filippo Bonchi, University of Pisa, Italy
Marcello Bonsangue, Leiden University, The Netherlands
Corina Cîrstea, University of Southampton, United Kingdom
Andrea Corradini, University of Pisa, Italy
José Fiadeiro, University of Dundee, United Kingdom
Ichiro Hasuo, National Institute of Informatics, Japan
Rolf Hennicker, Ludwig-Maximilians-Universität München, Germany
Bart Jacobs, Radboud University Nijmegen, The Netherlands
Bartek Klin, Warsaw University, Poland
Hans-Jörg Kreowski, Universität Bremen, Germany
Alexander Knapp, Universität Augsburg, Germany
Alexander Kurz, Chapman University, USA
Marina Lenisa, University of Udine, Italy
Stefan Milius (co-chair), Friedrich-Alexander Universität Erlangen-Nürnberg, Germany
Larry Moss, Indiana University, United States
Till Mossakowski, Otto-von-Guericke-Universität Magdeburg, Germany
Fernando Orejas, Technical University of Catalonia, Spain
Dirk Pattinson, Australian National University, Australia
Leila Ribeiro, Universidade Federal do Rio Grande do Sul, Brazil
Markus Roggenbach (co-chair), Swansea University, United Kingdom
Grigore Roşu, University of Illinois at Urbana-Champaign, United States
Alexandra Silva, University College London, United Kingdom

Program Committee

Filippo Bonchi, University of Pisa, Italy
Corina Cîrstea, University of Southampton, United Kingdom
Bob Coecke, University of Oxford, United Kingdom
José Luiz Fiadeiro, Royal Holloway University of London, United Kingdom
Daniel Gaina, Kyushu University, Japan
Sergey Goncharov, University of Erlangen-Nürnberg, Germany
Ichiro Hasuo, National Institute of Informatics, Japan
Chris Heunen, University of Edinburgh, United Kingdom
Helle Hvid Hansen, Delft University of Technology, The Netherlands
Magne Haveraaen, University of Bergen, Norway
Bart Jacobs, Radboud University Nijmegen, The Netherlands
Bartek Klin, Warsaw University, Poland
Alexander Knapp, Universität Augsburg, Germany
Ekaterina Komendantskaya, Heriot-Watt University, United Kingdom
Barbara König, Universität Duisburg-Essen, Germany
Clemens Kupke, University of Strathclyde, United Kingdom
Alexander Kurz, Chapman University, US
Narciso Martí-Oliet, Universidad Complutense Madrid, Spain

8th Conference on Algebra and Coalgebra in Computer Science (CALCO 2019).
Editors: Markus Roggenbach and Ana Sokolova



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

0:x **Conference Organization**

Larry Moss, Indiana University, US
Till Mossakowski, Otto-von-Guericke-Universität Magdeburg, Germany
Peter Ölveczky, University of Oslo, Norway
Dirk Pattinson, Australian National University, Australia
Daniela Petrisan, University Paris Diderot, France
Carlos Gustavo Lopez Pombo, Universidad de Buenos Aires, Argentina
Damien Pous, CNRS, France
Markus Roggenbach (PC co-chair), Swansea University, United Kingdom
Jurriaan Rot, Radboud University Nijmegen, The Netherlands
Pierre-Yves Schobbens, University of Namur, Belgium
Lutz Schröder, Friedrich-Alexander Universität Erlangen-Nürnberg, Germany
Ana Sokolova (PC co-chair), University of Salzburg, Austria
Ionuț Țuțu, Royal Holloway, United Kingdom / Romanian Academy, Romania
Fabio Zanasi, University College London, United Kingdom

Additional Reviewers

Mihai Codescu
David Frutos Escrig
Brendan Fong
Lorenzo Gheri
Håkon Gylterud
Gerco van Heerdt
Alexandre Madeira
Dan Marsden
Robin Piedeleu
John Power
David Sprunger
Lars Stoltenow
Henning Urbat
Niels van der Weide
Uwe Wolter

Matching μ -Logic: Foundation of \mathbb{K} Framework

Xiaohong Chen

University of Illinois at Urbana-Champaign, USA
<http://fs1.cs.illinois.edu/~xchen>
xc3@illinois.edu

Grigore Roşu

University of Illinois at Urbana-Champaign, USA
<http://fs1.cs.illinois.edu/~grosu>
grosu@illinois.edu

Abstract

\mathbb{K} framework is an effort in realizing the ideal language framework where programming languages must have formal semantics and all language tools are automatically generated from the formal semantics in a correct-by-construction manner at no additional costs. In this extended abstract, we present matching μ -logic as the foundation of \mathbb{K} and discuss some of its applications in defining constructors, transition systems, modal μ -logic and temporal logic variants, and reachability logic.

2012 ACM Subject Classification Theory of computation \rightarrow Logic

Keywords and phrases Matching μ -logic, Program verification, Reachability logic

Digital Object Identifier 10.4230/LIPIcs.CALCO.2019.1

Category Invited Paper

1 Introduction

In an ideal language framework, all programming languages must have formal semantics and all language tools are automatically generated from the formal semantics in a correct-by-construction manner at no additional costs. \mathbb{K} framework (www.kframework.org) is an almost 20-year continuous effort in realizing the ideal language framework. Many real-world languages such as C [5], Java [1], JavaScript [9] as well as the emerging blockchain languages such as EVM [6], have had their formal semantics successfully defined in \mathbb{K} and language tools such as parsers, interpreters, and deductive verifiers have been automatically generated by \mathbb{K} .

In terms of program verification, \mathbb{K} adopts a language-independent approach that is different from the classic language-specific approaches such as Hoare-style verification [7], where different languages have different program logics and thus different verifiers. Instead, the current \mathbb{K} implementation uses *matching logic* [10] to specify static structures of programs and *reachability logic* [11] to reason about dynamic reachability properties for all languages. Formal semantics are given as *theories* in these logics, so their fixed and thus language-independent proof systems achieve semantic-based program verification for all languages [4].

As its name suggests, reachability logic can only express reachability properties, which limits \mathbb{K} to verifying, for instance, liveness properties, which are beyond reachability logic but can naturally be expressed in temporal logics such as linear temporal logic (LTL) or computation tree logic (CTL). To overcome this limitation, we recently proposed *matching μ -logic* [2], which is a powerful logic that subsumes not only matching logic and reachability logic, but also first-order logic with least fixpoints, modal μ -logic, many variants of temporal logics, dynamic logic, and others (see Fig. 1). This demonstrates that matching μ -logic can serve as the uniform foundation of an ideal language framework.

Here we only present matching μ -logic by examples and show its application in specifying and reasoning about constructors, transition systems, and reachability. For more details see [2, 3].



© Xiaohong Chen and Grigore Roşu;

licensed under Creative Commons License CC-BY

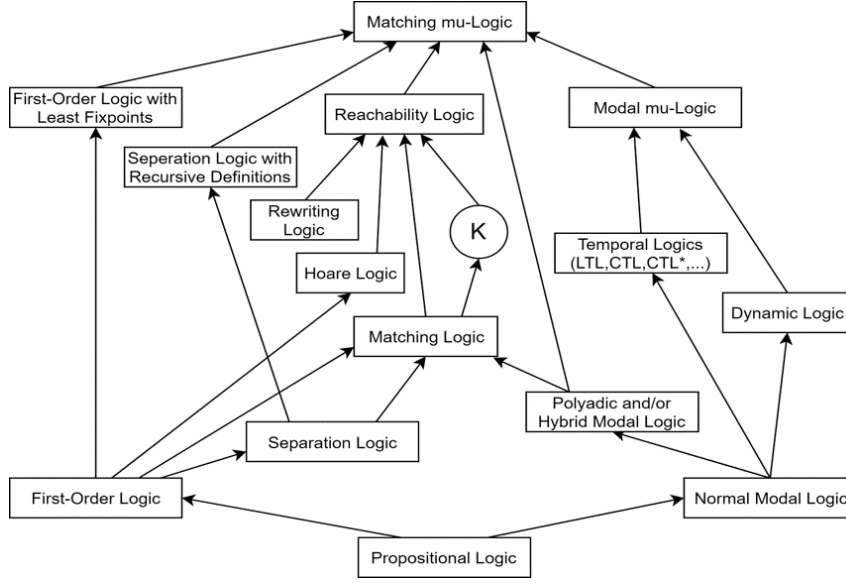
8th Conference on Algebra and Coalgebra in Computer Science (CALCO 2019).

Editors: Markus Roggenbach and Ana Sokolova; Article No. 1; pp. 1:1–1:4

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** Many popular logics can be defined in matching μ -logic as theories and notations [2]; the current \mathbb{K} implementation (denoted as the node labeled “K”) is so far the best effort in implementing reachability logic reasoning and will eventually be lifted to the same level as matching μ -logic.

2 Matching μ -Logic Examples

Preliminaries and basic examples

Matching logic (the version without μ) is a variant of many-sorted first-order logic (FOL) which makes no distinction between functions and predicates but uses *symbols* to uniformly build *patterns* that can represent static structures, dynamic properties, and logic constraints. Matching μ -logic extends matching logic with the least fixpoint μ -binder as in modal μ -logic [8], which can build *inductive patterns* to represent inductive and co-inductive data structures and recursive properties and logical constraints.

Intuitively speaking, a pattern evaluates to the set of elements *matching* it. For example:

- x , called an *element variable*, is matched by exactly one element x ;
- X , called a *set variable*, is matched by any set X of elements;
- $\text{succ}(x)$ is matched by the successor(s) of x ; here succ is a *symbol* that builds structures;
- $\exists x. \text{succ}(x)$ is matched by the successor of *some* x , i.e., all successors;
- $\text{zero} \vee \exists x. \text{succ}(x)$ is matched by either zero or successors;
- $\top \equiv \exists x. x$ is matched by x for some x , i.e., everything; $\perp \equiv \neg \top$ is matched by nothing;
- $\text{list}(x)$ is matched by all linked lists in the heap starting at pointer x ; list is also a symbol;
- $\text{list}(x) \wedge \text{prime}(x)$, same as above but with prime x ;
- $\mu N. \text{zero} \vee \text{succ}(N)$ is matched by all natural numbers zero , $\text{succ}(\text{zero})$, $\text{succ}(\text{succ}(\text{zero}))$, ...; this is because the μ -binder denotes the least set N w.r.t. set containment such that $N = \text{zero} \vee \text{succ}(N)$; in other words, N is the least set closed under zero and succ .

Constructors

The last example above $\mu N. \text{zero} \vee \text{succ}(N)$ that is matched by all natural numbers can be easily generalized to deal with any constructor set $C = \{c_i \mid c_i \text{ is a constructor of arity } n_i\}$, where the pattern $\mu D. \bigvee_{c_i \in C} c_i(\underbrace{D, \dots, D}_{n_i \text{ times}})$ evaluates to the least set that is closed under all constructors in C , yielding the set of all terms generated by C .

Transition systems and temporal logics

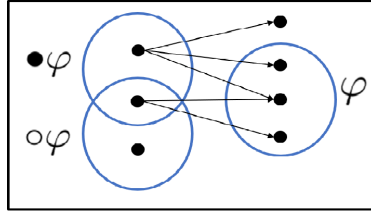
A transition system (S, R) is a pair of a state set S and a transition relation $R \subseteq S \times S$. In matching μ -logic, transition systems can be captured by one unary symbol \bullet called *one-path next* (we write $\bullet\varphi$ instead of $\bullet(\varphi)$) with the intended interpretation that $\bullet\varphi$ is matched by all *predecessors* of those matching φ :

$$\begin{array}{ccccccc} \dots & s & \xrightarrow{R} & s' & \xrightarrow{R} & s'' & \dots & // \text{ states} \\ & \bullet\bullet\varphi & & \bullet\varphi & & \varphi & & // \text{ patterns} \end{array}$$

In other words, a state matches $\bullet\varphi$ iff it has *one next* state that matches φ . Its dual *all-path next* $\circ\varphi \equiv \neg\bullet\neg\varphi$ is matched by those states whose *next states all* match φ (see Fig. 2).

We can define patterns that represent more complex dynamic properties. For example,

- $\bullet\top$ is matched by all non-terminal states;
- $\circ\perp$ is matched by all terminal states;
- $\diamond\varphi \equiv \mu X. \varphi \vee \bullet X$ is matched by all states that eventually reach φ on some path;
- $\square\varphi \equiv \nu X. \varphi \wedge \circ X$ is matched by all states that always stay in φ on all paths; ν -binder is the dual of μ -binder that builds greatest fixpoints instead of least fixpoints, defined as usual: $\nu X. \varphi \equiv \neg\mu X. \neg\varphi[\neg X/X]$ where $_[_/_]$ is the standard capture-avoiding substitution;
- $\text{WF} \equiv \mu X. \circ X$ is matched by all states that are well-founded, i.e., have no infinite paths.



■ **Figure 2** One/All-path next.

We point out that the above definitions are standard definitions in modal μ -logic. Since, as is well known, modal μ -logic subsumes many variants of temporal logic such as LTL and CTL and that matching μ -logic subsumes modal μ -logic (see [2, Section VII]), there is no surprise that matching μ -logic also subsumes LTL and CTL. What is interesting is that it only requires a few natural and intuitive axioms to faithfully capture LTL and CTL in matching μ -logic, as summarized below:

| Target logic | Assumption on traces | Axioms required in matching μ -logic |
|--------------------|----------------------------|--|
| Modal μ -logic | Any traces, no assumptions | No axioms |
| Infinite-trace LTL | Infinite and linear traces | (INF) + (LIN) |
| Finite-trace LTL | Finite and linear traces | (FIN) + (LIN) |
| CTL | Infinite traces | (INF) |

where (INF) is the pattern/axiom $\bullet\top$ stating that all states are non-terminal states, (FIN) is the pattern/axiom $\text{WF} \equiv \mu X. \circ X$ stating that all states are well-founded, and (LIN) is the pattern/axiom $\bullet X \rightarrow \circ X$ enforcing the linear paths: X holds on one next state implies X holds on all next states.

In conclusion, modal μ -logic is the empty theory over a unary symbol \bullet that contains no axioms. Adding (INF) yields precisely CTL. Adding (INF) yields precisely infinite-trace LTL and replacing (INF) with (FIN) yields finite-trace LTL. Therefore, matching μ -logic over the one-path next symbol \bullet gives a playground for defining variants of temporal logics.

Reachability logic

Our last example is to define reachability properties $\varphi \Rightarrow \varphi'$, called *reachability rules* [11], in matching μ -logic using the one-path next symbol. Here, φ and φ' are matching logic patterns not containing μ that are matched by program configurations. The semantics of $\varphi \Rightarrow \varphi'$ is that for every configuration γ that matches φ , either it reaches some configuration γ' that matches φ' in finitely many steps, or it is not well-founded. In other words, reachability is like a “weak” eventuality statement that applies to only well-founded states. This suggests to define the derived construct “weak eventually” $\Diamond_w \psi \equiv \nu X . \psi \vee \bullet X$, which is like the definition of the normal eventually $\Diamond \psi$ but replacing μ by ν , and define $\varphi \Rightarrow \varphi' \equiv \varphi \rightarrow \Diamond_w \varphi'$. We can prove that $\Diamond_w \psi = \Diamond \psi \vee \neg \text{WF}$, i.e., it indeed captures the semantics of (partial correctness) reachability, and thus our definition of reachability logic is faithful.

3 Conclusion

In this extended abstract, we presented matching μ -logic as the foundation of \mathbb{K} and discussed some of its applications to defining constructors, transition systems, modal μ -logic and temporal logic variants, and finally reachability logic.

References

- 1 Denis Bogdănaş and Grigore Roşu. K-Java: A complete semantics of Java. In *Proceedings of the 42nd Symposium on Principles of Programming Languages (POPL'15)*, pages 445–456. ACM, 2015.
- 2 Xiaohong Chen and Grigore Roşu. Matching μ -logic. In *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'19)*, 2019.
- 3 Xiaohong Chen and Grigore Roşu. Matching μ -logic. Technical report, University of Illinois at Urbana-Champaign, 2019. URL: <http://hdl.handle.net/2142/102281>.
- 4 Andrei Ştefănescu, Daejun Park, Shijiao Yuwen, Yilong Li, and Grigore Roşu. Semantics-based program verifiers for all languages. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'16)*, pages 74–91. ACM, 2016.
- 5 Chris Hathhorn, Chucky Ellison, and Grigore Roşu. Defining the undefinedness of C. In *Proceedings of the 36th annual ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'15)*, pages 336–345. ACM, 2015.
- 6 Everett Hildenbrandt, Manasvi Saxena, Xiaoran Zhu, Nishant Rodrigues, Philip Daian, Dwight Guth, Brandon Moore, Yi Zhang, Daejun Park, Andrei Ştefănescu, and Grigore Roşu. KEVM: A complete semantics of the Ethereum virtual machine. In *Proceedings of the 2018 IEEE Computer Security Foundations Symposium (CSF'18)*. IEEE, 2018. URL: <http://jellopaper.org>.
- 7 C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.
- 8 Dexter Kozen. Results on the propositional μ -calculus. In *Proceedings of the 9th International Colloquium on Automata, Languages and Programming (ICALP'82)*, pages 348–359. Springer, 1982.
- 9 Daejun Park, Andrei Ştefănescu, and Grigore Roşu. KJS: A complete formal semantics of JavaScript. In *Proceedings of the 36th annual ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'15)*, pages 346–356. ACM, 2015.
- 10 Grigore Roşu. Matching logic. *Logical Methods in Computer Science*, 13(4):1–61, 2017.
- 11 Grigore Roşu, Andrei Ştefănescu, Ştefan Ciobăcă, and Brandon M. Moore. One-path reachability logic. In *Proceedings of the 28th Symposium on Logic in Computer Science (LICS'13)*, pages 358–367. IEEE, 2013.

From Equational Specifications of Algebras with Structure to Varieties of Data Languages

Stefan Milius

Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
mail@stefan-milius.eu

Abstract

This extended abstract first presents a new category theoretic approach to equationally axiomatizable classes of algebras. This approach is well-suited for the treatment of algebras equipped with additional computationally relevant structure, such as ordered algebras, continuous algebras, quantitative algebras, nominal algebras, or profinite algebras. We present a generic HSP theorem and a sound and complete equational logic, which encompass numerous flavors of equational axiomizations studied in the literature. In addition, we use the generic HSP theorem as a key ingredient to obtain Eilenberg-type correspondences yielding algebraic characterizations of properties of regular machine behaviours. When instantiated for orbit-finite nominal monoids, the generic HSP theorem yields a crucial step for the proof of the first Eilenberg-type variety theorem for data languages.

2012 ACM Subject Classification Theory of computation → Equational logic and rewriting; Theory of computation → Formal languages and automata theory

Keywords and phrases Birkhoff theorem, Equational logic, Eilenberg theorem, Data languages

Digital Object Identifier 10.4230/LIPIcs.CALCO.2019.2

Category Invited Paper

Funding Supported by Deutsche Forschungsgemeinschaft (DFG) under project MI 717/5-2.

1 Equations and Algebras with Structure

A key tool in the algebraic theory of data structures is their specification by operations (constructors) and equations that they ought to satisfy. Birkhoff’s celebrated HSP theorem [7] states that a class of algebras over a signature Σ is a *variety* (i.e. closed under homomorphic images, subalgebras, and products) iff it is axiomatizable by equations $s = t$ between Σ -terms. Birkhoff also introduced a complete deduction system for reasoning about equations.

In algebraic approaches to the semantics of programming languages and computational effects, it is often natural to study algebras whose underlying sets are equipped with additional computationally relevant structure and whose operations preserve that structure. An important line of research thus concerns extensions of Birkhoff’s theory of equational axiomatization beyond ordinary Σ -algebras. On the syntactic level, this requires to enrich Birkhoff’s notion of an equation in ways that reflect the extra structure. For example, Bloom [8] and Adámek et al. [1, 2] established versions of the HSP theorem for *ordered algebras* and *continuous* ones, respectively, along with complete deduction systems. Here, the role of equations $s = t$ is taken over by inequations $s \leq t$. Recently, Mardare, Panangaden and Plotkin [19, 20] presented an HSP theorem for *quantitative algebras* and a complete deduction system. In the quantitative setting, equations $s =_{\epsilon} t$ are equipped with a non-negative real number ϵ , interpreted as “ s and t have distance at most ϵ ”. Varieties of *nominal algebras* were studied by Gabbay [15] and Kurz and Petrişan [18]. Here, the appropriate syntactic concept involves equations $s = t$ with constraints on the support of their variables. Finally, Reiterman [29] as well as Eilenberg and Schützenberger [13] showed that *pseudovarieties* (i.e. classes of *finite* algebras closed under homomorphic images, subalgebras and *finite*



© Stefan Milius;

licensed under Creative Commons License CC-BY

8th Conference on Algebra and Coalgebra in Computer Science (CALCO 2019).

Editors: Markus Roggenbach and Ana Sokolova; Article No. 2; pp. 2:1–2:5

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

products) can be axiomatized by so-called *profinite equations* or, equivalently, by sequences of ordinary equations $(s_i = t_i)_{i < \omega}$, interpreted as “all but finitely many of the equations $s_i = t_i$ hold”.

We propose a general category theoretic framework that allows to study equationally specified classes of algebras with extra structure in a systematic way. Our overall goal is to isolate the domain-specific part of any theory of equational axiomatization from its generic core. Our framework is parametric in the following data:

- a category \mathcal{A} with a factorization system $(\mathcal{E}, \mathcal{M})$;
- a full subcategory $\mathcal{A}_0 \subseteq \mathcal{A}$;
- a class Λ of cardinal numbers;
- a class $\mathcal{X} \subseteq \mathcal{A}$ of objects.

Here, \mathcal{A} is the category of algebras under consideration (e.g. ordered algebras, quantitative algebras, nominal algebras). Varieties are formed within \mathcal{A}_0 , and the cardinal numbers in Λ determine the arities of products under which the varieties are closed. Thus, the choice $\mathcal{A}_0 = \text{finite algebras}$ and $\Lambda = \text{finite cardinals}$ corresponds to pseudovarieties, and $\mathcal{A}_0 = \mathcal{A}$ and $\Lambda = \text{all cardinals}$ to varieties. The crucial ingredient of our setting is the parameter \mathcal{X} , which is the class of objects over which equations are formed. Typically, \mathcal{X} is chosen to be some class of freely generated algebras in \mathcal{A} . Equations are modeled as \mathcal{E} -quotients $e: X \twoheadrightarrow E$ (more generally, filters of such quotients) with domain $X \in \mathcal{X}$.

The choice of \mathcal{X} reflects the desired expressivity of equations in a given setting, and it determines the type of quotients under which equationally axiomatizable classes are closed. More precisely, in our category theoretic framework a *variety* is defined to be a subclass of \mathcal{A}_0 closed under $\mathcal{E}_{\mathcal{X}}$ -quotients, \mathcal{M} -subobjects, and Λ -products, where $\mathcal{E}_{\mathcal{X}}$ is a subclass of \mathcal{E} derived from \mathcal{X} . Due to its parametric nature, this concept of a variety is widely applicable and turns out to specialize to many interesting cases. The main result is the

► **General HSP Theorem** [22]. *A subclass of \mathcal{A}_0 forms a variety if and only if it is axiomatizable by equations.*

In addition, we introduce a generic deduction system for equations, based on two simple proof rules for equations $e: X \twoheadrightarrow E$, and establish a

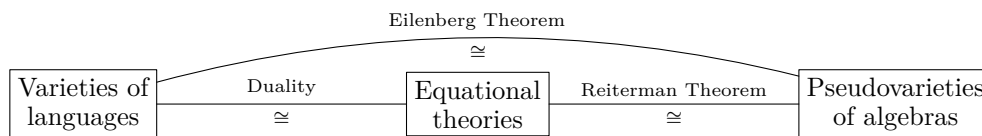
► **General Completeness Theorem** [22]. *The generic deduction system for equations is sound and complete.*

The above two theorems can be seen as the generic building blocks of the model theory of algebras with structure. They form the common core of numerous Birkhoff-type results and give rise to a systematic recipe for deriving concrete HSP and completeness theorems.

2 Varieties of Data Languages

Since the above results also cover Reiterman-type results (via the choice of \mathcal{X} as free algebras over *finite* sets) the General HSP Theorem yields a key tool for a generic algebraic language theory. In this theory one studies formal languages and other types of behaviours of finite machines (e.g. weighted languages, infinite words, trees, cost functions) in terms of algebraic structures that recognize them. As a prime example, regular languages can be described purely algebraically as the languages recognized by finite monoids, and a celebrated result by McNaughton, Papert, and Schützenberger [21, 33] asserts that a regular language is definable in first-order logic if and only if its syntactic monoid is aperiodic (i.e. it satisfies the equation $x^{n+1} = x^n$ for sufficiently large n). As an immediate application, this algebraic

characterization yields an effective procedure for deciding first-order definability. The first systematic approach to correspondence results of this kind was initiated by Eilenberg [14] who proved that *varieties of languages* (i.e. classes of regular languages closed under the set-theoretic boolean operations, derivatives, and homomorphic preimages) correspond bijectively to pseudovarieties of monoids. Inspired by Eilenberg's work, over the past four decades numerous further variety theorems were discovered for regular languages [16, 24, 27, 36], treating notions of varieties with modified closure properties, but also for machine behaviors beyond finite words, including weighted languages over a field [30], infinite words [25, 38], words on linear orderings [5, 6], ranked trees [4], binary trees [32], and cost functions [12]. Recent research [3, 9] has focused on generic approaches and has culminated in Salamanca's work [31] and our General Eilenberg Theorem that covers all of the above ones as special instances [37]. Its proof is based on two key ingredients: (1) *duality* in order to establish a correspondence between (profinite) equational theories and varieties of recognizable languages and (2) a generic *Reiterman-type correspondence* to pseudovarieties.



That duality plays an important role for Eilenberg-type correspondences has been established by Gehrke, Grigorieff and Pin [16]. The duality based proofs in [31, 37] yield a blueprint for new correspondences of this kind. For example, it allows to obtain the first Eilenberg-type correspondence for *data languages* [22]. Such languages are of significant interest in recent years, driven by practical applications in various areas of computer science, including efficient parsing of XML documents or software verification. Mathematically, data languages are modeled using *nominal sets* (see e.g. [26]). Over the years, several machine models for handling data languages of different expressive power have been proposed; see [34, 35] for a comprehensive survey. Here we focus on languages recognized by *orbit-finite nominal monoids*. They form an important subclass of the languages accepted by Francez and Kaminski's *finite memory automata* [17] (which are expressively equivalent to orbit-finite automata in the category of nominal sets [11]) and have been characterized by a fragment of monadic second-order logic over data words called *rigidly guarded MSO* [28]. In addition, Bojańczyk [10] and Colcombet, Ley and Puppis [28] established nominal versions of the McNaughton-Papert-Schützenberger theorem and showed that the first-order definable data languages are precisely the ones recognizable by aperiodic orbit-finite monoids. It is therefore natural to ask whether an Eilenberg-type theorem can be developed for data languages, and we answer this question affirmatively:

► **Nominal Eilenberg Theorem** [23]. *Varieties of data languages correspond bijectively to pseudovarieties of nominal monoids.*

Here, the notion of a *pseudovariety of nominal monoids* is as expected: a class of orbit-finite nominal monoids closed under quotient monoids, submonoids, and finite products. In contrast, the notion of a *variety of data languages* requires two extra conditions unfamiliar from other Eilenberg-type correspondences, most notably a technical condition called *completeness*. Like the original Eilenberg theorem, its nominal version gives rise to a generic relation between properties of data languages and properties of nominal monoids. For instance, the aperiodic orbit-finite monoids form a pseudovariety, and the first-order definable data languages form a variety, and thus the equivalence of these concepts can be understood as an instance of the nominal Eilenberg correspondence.

It should be pointed out that the Nominal Eilenberg Theorem requires new techniques and cannot be obtained as a mere instance of the previous General Eilenberg Theorem, since the latter is based on working with algebraic-like base categories (which excludes nominal sets) and the recognition by finite structures. However, our approach can be seen as an indication of the robustness of the key ideas behind the duality-based methodology for algebraic recognition and the guidance they provide towards future applications and results.

References

- 1 J. Adámek, A. H. Mekler, E. Nelson, and J. Reiterman. On the logic of continuous algebras. *Notre Dame J. Formal Logic*, 29(3):365–380, 1988.
- 2 Jiří Adámek, Evelyn Nelson, and Jan Reiterman. The Birkhoff Variety Theorem for continuous algebras. *Algebra Universalis*, 20(3):328–350, 1985.
- 3 Jiří Adámek, Stefan Milius, Robert S.R. Myers, and Henning Urbat. Generalized Eilenberg Theorem: Varieties of Languages in a Category. *ACM Trans. Comput. Log.*, 20(1):3:1–3:47, 2019.
- 4 J. Almeida. On pseudovarieties, varieties of languages, filters of congruences, pseudoidentities and related topics. *Algebra Universalis*, 27(3):333–350, 1990.
- 5 N. Bedon and O. Carton. An Eilenberg theorem for words on countable ordinals. In *Proc. LATIN’98*, volume 1380 of *LNCS*, pages 53–64. Springer, 1998.
- 6 N. Bedon and C. Rispal. Schützenberger and Eilenberg theorems for words on linear orderings. In *Proc. DLT’05*, volume 3572 of *LNCS*, pages 134–145. Springer, 2005.
- 7 Garret Birkhoff. On the structure of abstract algebras. *Proceedings of the Cambridge Philosophical Society*, 10:433—454, 1935.
- 8 Stephen L. Bloom. Varieties of ordered algebras. *J. Comput. Syst. Sci.*, 2(13):200–212, 1976.
- 9 M. Bojańczyk. Recognisable languages over monads. In I. Potapov, editor, *Proc. DLT’15*, volume 9168 of *LNCS*, pages 1–13. Springer, 2015. [arXiv:1502.04898](https://arxiv.org/abs/1502.04898).
- 10 Mikołaj Bojańczyk. Nominal Monoids. *Theory of Computing Systems*, 53(2):194–222, 2013.
- 11 Mikołaj Bojańczyk, Bartek Klin, and Sławomir Lasota. Automata theory in nominal sets. *Log. Methods Comput. Sci.*, 10(3:4):44 pp., 2014.
- 12 L. Daviaud, D. Kuperberg, and J.-É. Pin. Varieties of Cost Functions. In N. Ollinger and H. Vollmer, editors, *Proc. STACS 2016*, volume 47 of *LIPICs*, pages 30:1–30:14. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2016.
- 13 S. Eilenberg and M. P. Schützenberger. On pseudovarieties. *Advances Math.*, 10:413–418, 1976.
- 14 Samuel Eilenberg. *Automata, Languages, and Machines Vol. B*. Academic Press, 1976.
- 15 Murdoch J. Gabbay. Nominal algebra and the HSP theorem. *Journal of Logic and Computation*, 19:341–367, 2009.
- 16 Mai Gehrke, Serge Grigorieff, and Jean-Éric Pin. Duality and equational theory of regular languages. In *Proc. ICALP’08, Part II*, volume 5126 of *LNCS*, pages 246–257. Springer, 2008.
- 17 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theoret. Comput. Sci.*, 134(2):329–363, 1994.
- 18 Alexander Kurz and Daniela Petrisan. On universal algebra over nominal sets. *Mathematical Structures in Computer Science*, 20(2):285–318, 2010.
- 19 Radu Mardare, Prakash Panangaden, and Gordon Plotkin. Quantitative Algebraic Reasoning. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS ’16*, pages 700–709. ACM, 2016.
- 20 Radu Mardare, Prakash Panangaden, and Gordon Plotkin. On the axiomatizability of quantitative algebras. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017. [doi:10.1109/LICS.2017.8005102](https://doi.org/10.1109/LICS.2017.8005102).

- 21 Robert McNaughton and Seymour A. Papert. *Counter-Free Automata (M.I.T. Research Monograph No. 65)*. The MIT Press, 1971.
- 22 Stefan Milius and Henning Urbat. Equational Axiomatization of Algebras with Structure. In Mikołaj Bojańczyk and Alex Simpson, editors, *Proc. Foundations of Software Science and Computation Structures (FoSSaCS)*, volume 11425 of *Lecture Notes Comput. Sci. (ARCoSS)*, pages 400–417, 2019.
- 23 Stefan Milius and Henning Urbat. Varieties of Data Languages. In *Proc. 46th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 132 of *LIPICs*, pages 130:1–130:14, 2019. (Full version available online at [arXiv:1903.08053](https://arxiv.org/abs/1903.08053)). doi:10.4230/LIPICs.ICALP.2019.130.
- 24 J.-É. Pin. A variety theorem without complementation. *Russ. Math.*, 39:80–90, 1995.
- 25 J.-É. Pin. Positive varieties and infinite words. In *LATIN 98*, volume 1380 of *LNCS*, pages 76–87. Springer, 1998.
- 26 Andrew M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge University Press, 2013.
- 27 L. Polák. Syntactic semiring of a language. In J. Sgall, A. Pultr, and P. Kolman, editors, *Proc. MFCS'01*, volume 2136 of *LNCS*, pages 611–620. Springer, 2001.
- 28 Gabriele Puppis, Thomas Colcombet, and Clemens Ley. Logics with rigidly guarded data tests. *Log. Methods Comput. Sci.*, 11(3:10):56 pp., 2015.
- 29 J. Reiterman. The Birkhoff theorem for finite algebras. *Algebra Universalis*, 14(1):1–10, 1982.
- 30 C. Reutenauer. Séries formelles et algèbres syntactiques. *J. Algebra*, 66:448–483, 1980.
- 31 Julian Salamanca. Unveiling Eilenberg-type Correspondences: Birkhoff’s Theorem for (finite) Algebras + Duality, February 2017. [arXiv:1702.02822](https://arxiv.org/abs/1702.02822).
- 32 S. Salehi and M. Steinby. Tree algebras and varieties of tree languages. *Theor. Comput. Sci.*, 377(1-3):1–24, 2007.
- 33 Marcel-Paul Schützenberger. On finite monoids having only trivial subgroups. *Inform. and Control*, 8:190–194, 1965.
- 34 Thomas Schwentick. Automata for XML – A survey. *J. Comput. System Sci.*, 73(3):289–315, 2007.
- 35 Luc Segoufin. Automata and Logics for Words and Trees over an Infinite Alphabet. In *Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006, Proceedings*, pages 41–57, 2006.
- 36 H. Straubing. On logical descriptions of regular languages. In S. Rajsbaum, editor, *LATIN 2002 Theor. Informatics*, volume 2286 of *LNCS*, pages 528–538. Springer, 2002.
- 37 Henning Urbat, Jiří Adámek, Liang-Ting Chen, and Stefan Milius. Eilenberg Theorems for Free. In Kim G. Larsen, Hans L. Bodlaender, and Jean-François Raskin, editors, *Proc. 42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017)*, volume 83 of *LIPICs*. Schloss Dagstuhl, 2017. EATCS Best Paper Award.
- 38 T. Wilke. An Eilenberg Theorem for ∞ -Languages. In *Proc. ICALP'91*, volume 510 of *LNCS*, pages 588–599. Springer, 1991.

Principles of Natural Language, Logic, and Tensor Semantics

Mehrnoosh Sadrzadeh

School of Electronic Engineering and Computer Science, Queen Mary University of London, UK
mehrnoosh.sadrzadeh@qmul.ac.uk

Abstract

Residuated monoids model the structure of sentences. Vectors provide meaning representations for words. A functorial mapping between the two is obtained by lifting the vectors to tensors. The resulting sentence representations solve similarity, disambiguation and entailment tasks.

2012 ACM Subject Classification Theory of computation → Categorical semantics; Computing methodologies → Natural language processing; General and reference → Experimentation

Keywords and phrases Residuated Monoids, Vector Space Semantics, Corpora of Textual Data, Sentence Similarity and Disambiguation

Digital Object Identifier 10.4230/LIPIcs.CALCO.2019.3

Category Invited Paper

Funding Royal Academy of Engineering Industrial Fellowship Scheme *IFS1718\63*, Royal Society International Exchange Award *IE161631*

1 The Algebra of Grammatical Types

A partially ordered monoid is called *residuated* and is denoted by $(M, \cdot, 1, \leq, \rightarrow, \leftarrow)$, whenever for $b, c \in M$ we have $c \cdot c \rightarrow b \leq b$ and $b \leftarrow c \cdot c \leq b$. Given a set of basic types \mathcal{B} and a vocabulary Σ , a *monoid grammar* is the tuple $(\mathcal{T}(\mathcal{B}), \Sigma, \mathcal{D}, \{s\})$, wherein $\mathcal{T}(\mathcal{B})$ is a residuated monoid generated over \mathcal{B} and $\mathcal{D} \subseteq \Sigma \times \mathcal{T}(\mathcal{B})$ is a type assignment to the vocabulary. A string of words $w_1 w_2 \cdots w_n$ is *grammatical* in a monoid grammar, whenever for $(w_i, t_i) \in \mathcal{D}$, we have $t_1 \cdot t_2 \cdots t_n \leq s$, where s is an element of \mathcal{B} and stands for the type of a sentence.

As an example, consider the vocabulary $\Sigma = \{\text{men, dogs, cute, kill}\}$ and the type dictionary $\mathcal{D} = \{(\text{men}, n), (\text{dogs}, n), (\text{cute}, n \leftarrow n), (\text{kill}, (n \rightarrow s) \leftarrow n)\}$. The sentence “men kill cute dogs” is grammatical, since we have

$$n \cdot ((n \rightarrow s) \leftarrow n) \cdot (n \leftarrow n) \cdot n \leq n \cdot ((n \rightarrow s) \leftarrow n) \cdot n \leq n \cdot (n \rightarrow s) \leq s$$

2 Tensor Semantics

Suppose W is a vector space with a set of fixed orthonormal basis $\{b_i\}_i$. Elements of W are vectors $\sum_i c_i b_i$ and elements of $\underbrace{W \otimes \cdots \otimes W}_n$ are tensors $T_{i_1 i_2 \cdots i_n} = \sum_{i_1 i_2 \cdots i_n} C_{i_1 i_2 \cdots i_n} b_{i_1} \otimes$

$b_{i_2} \otimes \cdots \otimes b_{i_n}$. The action of a tensor on another tensor is called *tensor contraction* and is defined as $T_{i_1 i_2 \cdots i_n} T_{i_n i_{n+1} \cdots i_{n+k}} = T_{i_1 i_2 \cdots i_{n+1} \cdots i_{n+k}} \in \underbrace{W \otimes \cdots \otimes W}_{n+k-1}$.

We develop a mapping \mathcal{F} between a monoid grammar and the tensor powers of W . To basic types $t \in \mathcal{B}$, we assign W , i.e., $\mathcal{F}(t) := W$; to words w with basic types t we assign elements of W , i.e., $\mathcal{F}(w) := T_i \in W$. To complex types, we assign tensors of W as follows

$$\mathcal{F}(t_1 \cdot t_2) = \mathcal{F}(t_1 \rightarrow t_2) = \mathcal{F}(t_1 \leftarrow t_2) := \mathcal{F}(t_1) \otimes \mathcal{F}(t_2)$$



© Mehrnoosh Sadrzadeh;

licensed under Creative Commons License CC-BY

8th Conference on Algebra and Coalgebra in Computer Science (CALCO 2019).

Editors: Markus Roggenbach and Ana Sokolova; Article No. 3; pp. 3:1–3:4

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Words with complex types are assigned elements of the tensor spaces of their types, that is, $\mathcal{F}(w) = T_{i_1 i_2 \dots i_n} \in \underbrace{W \otimes \dots \otimes W}_n$. Given a grammatical sentence $w_1 w_2 \dots w_n$, its tensor meaning is defined as the n tensor contraction of the tensor semantics of its words, that is, $\mathcal{F}(w_1 w_2 \dots w_n) := \mathcal{F}(w_1) \mathcal{F}(w_2) \dots \mathcal{F}(w_n)$.

As an example, suppose we assign vectors T_k^{dogs} and T_j^{men} in W to *men* and *dogs*, the matrix T_{lk}^{cute} in $W \otimes W$ to *cute* and the cube T_{ijk}^{kill} in $W \otimes W \otimes W$ to *kill*. The meaning of “men kill cute dogs” is computed via the following contraction of tensors $T_j^{\text{men}} T_{ijl}^{\text{kill}} T_{lk}^{\text{cute}} T_k^{\text{dogs}}$. Recall that when we have a fixed set of orthonormal basis, $T_{ij} \cong T_{ji}$.

3 Implementation on Corpora of Textual Data

Given a corpus of text, e.g. the English Wikipedia, a set of target words T and a set of context words C , a vector space W is created over C . In this vector space, each target word has a vector, where each c_i is (a function of) the number of times w occurred with each basis vector in a neighbourhood window, e.g. 5 words to the left or right. As an example, suppose $C = \{\text{blood, grave, dead}\}$ and $T = \{\text{vampire, zombie, butterfly}\}$ and the following vectors

$$T_i^{\text{zombie}} = (17, 13, 10) \quad T_i^{\text{vampire}} = (15, 9, 8) \quad T_i^{\text{butterfly}} = (0, 1, 3)$$

Words that have complex types are modelled as tensors. The tensors are learnt by first building vector representations for phrases containing the words, then *learning* a tensor whose contraction with the tensors of other words in the phrase provides a reasonable approximation for the vector of the phrase. For example, in order to learn a matrix for the adjective *green*, we first build vectors for all the adjective-noun phrases with *green* as adjective, e.g. for *green grass*, *green dress*, *green space*. Machine learning algorithms such as *least squared distance* are employed to learn an approximation for T_{ij}^{green} such that

$$T_i^{\text{green grass}} \sim T_{ij}^{\text{green}} T_j^{\text{grass}} \quad T_i^{\text{green dress}} \sim T_{ij}^{\text{green}} T_j^{\text{dress}} \quad T_i^{\text{green space}} \sim T_{ij}^{\text{green}} T_j^{\text{space}}$$

Once the grammatical structure of a language is modelled in a monoid grammar and word vectors and tensors have been built for its vocabulary, tensor contraction is applied to obtain vector representations for its sentences. The cosine distances between these representations provide a measure of sentence similarity and are applied to paraphrasing and disambiguation tasks. For paraphrasing, one builds vectors for sentences such as “man shut doors”, “gentleman closed eyes”, “programme faces difficulty”, “project hits problem” and uses their distances to decide that the latter two are more similar than the former two. For disambiguation, one builds vectors for sentences such as “man drew sword”, “man sketched sword”, “man pulled sword” to decide whether *drew* means *sketched* or *pulled*.

4 History and References

Similar to programming languages, natural languages have different characteristic features such as morphology, phonology, syntax, semantics, and pragmatics. Formal structures have been used to study these features and indeed ideas are shared between natural and programming semantics communities. An example is the setting of Context Free Grammars, introduced by Chomsky to analyse the grammatical structure of English [4] and subsequently applied to other languages and programming languages. The first algebraic approaches to natural language go back to the work of Ajdukiewicz [1], where structures similar to groups were used to provide a functional interpretation for grammatical types. These systems

were later refined with a noncommutative multiplication by Bar-Hillel [2] and then Lambek developed a residuated monoid semantics and a cut-free sequent calculus for them [15]. The expressive powers of these two systems were proven equivalent by Pentus [20].

The vector space semantics of natural language is motivated by the *distributional* semantic ideas of Firth [8] and Harris [12], who argued that words that occur in the same contexts have similar meanings. These models were both implemented in Information Retrieval [27] and applied to Natural Language Processing [24].

Encoding a model of grammar in vector space semantics to obtain vector representations for sentences was an open problem until recently. In 2007 Clark and Pulman showed how a context free parse tree of a sentence can be assigned a tensor semantics by taking the Kronecker products of the vectors of the words therein and the symbolic vectors of their grammatical roles [5]. It was not clear, however, how to build vectors for grammatical roles. Between 2008 and 2011, with Clark, Coecke, and Preller we showed that if one uses Lambek's pregroup grammars [16, 23] one obtains a functorial semantics in the compact closed category of finite dimensional vector spaces and linear maps [6, 22]. Later with Coecke and Grefenstette, we showed how residuated monoid grammars also get a functorial semantics via the translation between a residuated monoid and a pregroup [7]. More recently, I showed how one can get by without using category theory and still be able to express this semantics using the language of tensor contraction [25]; this is via the \mathcal{F} mapping that I have tried to spell out in this abstract.


Starting from 2011, we have implemented and experimented with the tensor models on large corpora of textual data in similarity, disambiguation, and entailment tasks and showed that in each case there is a tensor model that outperforms the vector models [10, 11, 13, 19, 14, 26]. The method that we have described here and which is used to learn the tensors was introduced by Baroni and Zamparelli for adjectives [3] and later extended to verbs [9, 21]. Maillard and Clark [17] showed how one can use neural networks and the Skipgram algorithm of Mikolov [18] to obtain much better results. In work in progress with Clark and Wijnholds, we are extending these models to arbitrary tensors.

References

- 1 K. Ajdukiewicz. Die syntaktische Konnexitat. *Studia Philosophica*, 1:1–27, 1935.
- 2 Y. Bar-Hillel. A quasi-arithmetical notation for syntactic description. *Language*, 29:47–58, 1953.
- 3 M. Baroni and R. Zamparelli. Nouns are vectors, adjectives are matrices: Representing adjective-noun constructions in semantic space. In *Conference on Empirical Methods in Natural Language Processing*, Cambridge, MA, 2010.
- 4 Noam Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2:113–124, 1956.
- 5 Stephen Clark and Stephen Pulman. Combining Symbolic and Distributional Models of Meaning. In *Proceedings of the AAAI Spring Symposium on Quantum Interaction*, pages 52–55, 2007.
- 6 B. Coecke, M. Sadrzadeh, and S. Clark. Mathematical Foundations for Distributed Compositional Model of Meaning. Lambek Festschrift. *Linguistic Analysis*, 36:345–384, 2010.
- 7 Bob Coecke, Edward Grefenstette, and Mehrnoosh Sadrzadeh. Lambek vs. Lambek: Functorial vector space semantics and string diagrams for Lambek calculus. *Annals of Pure and Applied Logic*, 164(11):1079–1100, 2013. Special issue on Seventh Workshop on Games for Logic and Programming Languages.
- 8 J.R. Firth. A Synopsis of Linguistic Theory 1930–1955. In *Studies in Linguistic Analysis*. Longmans, 1957.

- 9 E. Grefenstette, G. Dinu, Y. Zhang, M. Sadrzadeh, and M. Baroni. Multi-Step Regression Learning for Compositional Distributional Semantics. In *10th International Conference on Computational Semantics*, Postdam, 2013.
- 10 E. Grefenstette and M. Sadrzadeh. Experimental Support for a Categorical Compositional Distributional Model of Meaning. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*, pages 1394–1404, 2011.
- 11 Edward Grefenstette and Mehrnoosh Sadrzadeh. Concrete Models and Empirical Evaluations for the Categorical Compositional Distributional Model of Meaning. *Computational Linguistics*, 41:71–118, 2015.
- 12 Z.S. Harris. Distributional structure. *Word*, 1954.
- 13 D. Kartsaklis and M. Sadrzadeh. Prior Disambiguation of Word Tensors for Constructing Sentence Vectors. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*, 2013.
- 14 Dimitri Kartsaklis, Nal Kalchbrenner, and Mehrnoosh Sadrzadeh. Resolving Lexical Ambiguity in Tensor Regression Models of Meaning. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, Baltimore, MD, USA, Volume 2: Short Papers*, pages 212–217, 2014.
- 15 J. Lambek. The mathematics of sentence structure. *American Mathematics Monthly*, 65, 1958.
- 16 J. Lambek. Type grammars revisited. In *proceedings of LACL 97*, volume 1582 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, 1997.
- 17 Jean Maillard and Stephen Clark. Learning adjective meanings with a tensor-based skip-gram model. In *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*, pages 327–331, 2015.
- 18 Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.
- 19 Dmitrijs Milajevs, Dimitri Kartsaklis, Mehrnoosh Sadrzadeh, and Matthew Purver. Evaluating Neural Word Representations in Tensor-Based Compositional Settings. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 708–719. Association for Computational Linguistics, 2014.
- 20 Mati Pentus. Lambek Grammars Are Context Free. In *In Proceedings of the Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 429–433. IEEE Computer Society Press, 1993.
- 21 Tamara Polajnar, Luana Fagarasan, and Stephen Clark. Reducing dimensions of tensors in type-driven distributional semantics. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1036–1046, 2014.
- 22 A. Preller and M. Sadrzadeh. Bell States and Negative Sentences in the Distributed Model of Meaning. In P. Selinger B. Coecke, P. Panangaden, editor, *Electronic Notes in Theoretical Computer Science, Proceedings of the 6th Workshop on Quantum Physics and Logic*, volume 270, pages 141–153, 2010.
- 23 Anne Preller and Joachim Lambek. Free compact 2-categories. *Mathematical Structures in Computer Science*, 17:309–340, 2007.
- 24 H. Rubenstein and J.B. Goodenough. Contextual Correlates of Synonymy. *Communications of the ACM*, 8(10):627–633, 1965.
- 25 M. Sadrzadeh. Unifying the Mathematics of Natural Language Grammar and Data. *London Mathematical Society News Letter*, pages 25–31, 2018.
- 26 Mehrnoosh Sadrzadeh, Dimitri Kartsaklis, and Esmā Balkır. Sentence entailment in compositional distributional semantics. *Annals of Mathematics and Artificial Intelligence*, 82(4):189–218, 2018.
- 27 G. Salton, A. Wong, and C. S. Yang. A Vector Space Model for Automatic Indexing. *Commun. ACM*, 18:613–620, 1975.

Coinduction: Automata, Formal Proof, Companions

Damien Pous 

Univ Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP, Lyon, France

<http://perso.ens-lyon.fr/damien.pous/>

Damien.Pous@ens-lyon.fr

Abstract

Coinduction is a mathematical tool that is used pervasively in computer science: to program and reason about infinite data-structures, to give semantics to concurrent systems, to obtain automata algorithms. We present some of these applications in automata theory and in formalised mathematics. Then we discuss recent developments on the abstract theory of coinduction and its enhancements.

2012 ACM Subject Classification Theory of computation → Logic

Keywords and phrases Coinduction, Automata, Coalgebra, Formal proofs

Digital Object Identifier 10.4230/LIPIcs.CALCO.2019.4

Category Invited Paper

Funding This work has been funded by the European Research Council (ERC) under the European Union's Horizon 2020 programme (CoVeCe, grant agreement No 678157).

Induction and coinduction

Induction and coinduction are used in two main ways in computer science: to define datatypes and compute with those, and to define predicates and reason about them. The former can be presented using category theory: inductive datatypes such as natural numbers, lists, or trees can be modelled as initial algebras, while coinductive datatypes such as streams, infinite trees, automata, or labelled transitions systems can be modelled as final coalgebras. In contrast, we use lattice theory for predicates: inductive predicates such as reachability or derivability in a proof system are presented as least fixpoints while coinductive predicates such as divergence, bisimilarity, or language equivalence, are presented as greatest fixpoints.

When doing a proof by induction, one has to be careful about two things: 1/ choosing an appropriate induction principle (e.g., simple induction, rank-2 induction, or course of value induction), and 2/ choosing a strong enough invariant. We shall see that the very same observation applies with coinduction.

Automata algorithms

Take for instance algorithms for checking language equivalence of finite deterministic automata. Language equivalence can be characterised as a greatest fixpoint, so that it can be checked using a coinductive algorithm: start from a relation consisting of the pair of initial states, and widen this relation until it becomes a *bisimulation* [5]. This iterative process corresponds to point 2/ above: we iteratively refine an initial guess until we get a proper invariant. Such an algorithm can be made more efficient by choosing a more powerful coinduction principle, e.g., *bisimulations up to equivalence*, or *bisimulations up to congruence* for non-deterministic automata; this is point 1/ above.



© Damien Pous;

licensed under Creative Commons License CC-BY

8th Conference on Algebra and Coalgebra in Computer Science (CALCO 2019).

Editors: Markus Roggenbach and Ana Sokolova; Article No. 4; pp. 4:1–4:4

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Formal proofs and equational theories

Coinduction can be used for many automata algorithms, which in turn can be used in the context of formal proofs, to improve automation and delegate administrative steps to the computer. Indeed, several equational theories can be decided by characterising them in terms of language equivalence. The key example is that of *Kleene algebra*: this (quasi)equational theory admits binary relations and formal languages as free models [13, 15, 4], and is decidable in PSPACE using automata algorithms. Therefore, proof obligations corresponding to this fragment can be discharged automatically in proof assistants [6], using so-called *reflexive tactics*. Another important theory is that of Kleene algebra with tests (KAT) [14], which can be decided using automata on guarded strings. This was used successfully to reason about while programs and flowchart schemes in the Coq proof assistant [18].

Other theories include Kleene algebra with converse [3, 10], Kleene allegories [7, 17], and concurrent Kleene algebra [8]. Those respectively require working with downward-closed languages, languages of graphs, and languages of partially ordered multisets (pomsets) [20]. Decidability can be obtained by designing appropriate automata models, and by characterising language equivalence as a greatest fixpoint (a coinductive predicate). In some cases, completeness can be obtained by reasoning about this greatest fixpoint [9]. Those various results are technically involved, so that formalising them in a proof assistant in order to extend the aforementioned reflexive tactics would require an important work.

Theory of enhanced coinduction in complete lattices

For predicates, the theory of coinduction can be expressed in complete lattices, starting from Knaster-Tarski's theorem [12, 25]: every monotone function b in a complete lattice admits a greatest fixpoint νb . Enhancements, or up-to techniques, have been studied by Sangiorgi [23, 24]. Given a function b , the idea is to find other functions b' that are easier to use, and such that $\nu b' = \nu b$. A recent proposal [19] consists in using the function $b' = bt$, where t , the *companion* of b is the largest function f such that $fb \leq bf$. This simple idea greatly simplifies the whole theory: the companion is a closure operator and it intuitively contains all potential enhancements. It moreover makes it possible present coinductive proofs on the fly, without needing to announce the invariant upfront – a important feature when it comes of formalisation in proof assistants [11].

Theory of enhanced coinduction in category theory

Streams of real numbers form the final coalgebra for the functor $BX = \mathbb{R} \times X$. This observation makes it possible to define streams corecursively: it suffices to provide a coalgebra for B . The constant streams, the stream *nat* of natural numbers, and the pointwise addition of streams can be defined in this way. Note that for *nat*, one has to define a coalgebra that provides not only *nat*, but also all its suffixes; this is point 2/ above: one has to provide a strong enough invariant. In slightly more involved situations, one has to use a stronger corecursion principle (point 1/ above). This the case for the convolution product, whose natural definition builds on addition [22]. A standard solution [16, 1] consists in using *distributive laws* $\lambda : FB \Rightarrow BF$, and such techniques were recently implemented in Isabelle/HOL [2]. Given the aforementioned situation in complete lattices, one can naturally ask whether there exists a “largest” such distributive law, a *companion* T for B [21]. If it exists, the companion is a monad; if B preserves the codensity monad of its final sequence, then the companion is that codensity; this is the case for polynomial functors, and in this case the companion can be characterised in terms of *causal algebras* on the final coalgebra. An intriguing open question is whether the finite powerset functor admits a companion.

References

- 1 F. Bartels. *On generalised coinduction and probabilistic specification formats*. PhD thesis, CWI, Amsterdam, April 2004.
- 2 Jasmin Christian Blanchette, Aymeric Bouzy, Andreas Lochbihler, Andrei Popescu, and Dmitriy Traytel. Friends with Benefits - Implementing Corecursion in Foundational Proof Assistants. In *ESOP*, volume 10201 of *LNCS*, pages 111–140. Springer, 2017. doi:10.1007/978-3-662-54434-1_5.
- 3 S. L. Bloom, Z. Ésik, and G. Stefanescu. Notes on equational theories of relations. *Algebra Universalis*, 33(1):98–126, 1995. doi:10.1007/BF01190768.
- 4 Maurice Boffa. Une remarque sur les systèmes complets d'identités rationnelles. *Informatique Théorique et Applications*, 24:419–428, 1990. URL: http://archive.numdam.org/article/ITA_1990__24_4_419_0.pdf.
- 5 Filippo Bonchi and Damien Pous. Checking NFA equivalence with bisimulations up to congruence. In *POPL*, pages 457–468. ACM, 2013. doi:10.1145/2429069.2429124.
- 6 Thomas Braibant and Damien Pous. Deciding Kleene Algebras in Coq. *LMCS*, 8(1):1–16, 2012. doi:10.2168/LMCS-8(1:16)2012.
- 7 Paul Brunet and Damien Pous. Petri automata for Kleene allegories. In *LICS*, pages 68–79. ACM, 2015. doi:10.1109/LICS.2015.17.
- 8 Paul Brunet, Damien Pous, and Georg Struth. On decidability of Concurrent Kleene Algebra. In *CONCUR*, volume 85 of *LIPICs*, pages 24:1–24:16. Schloss Dagstuhl, 2017. doi:10.4230/LIPICs.CONCUR.2017.28.
- 9 Amina Doumane and Damien Pous. Completeness for identity-free Kleene Lattices. In *CONCUR*, volume 118 of *LIPICs*, pages 18:1–18:17. Schloss Dagstuhl, 2018. doi:10.4230/LIPICs.CONCUR.2018.18.
- 10 Z. Ésik and L. Bernátsky. Equational properties of Kleene algebras of relations with conversion. *Theoretical Computer Science*, 137(2):237–251, 1995. doi:10.1016/0304-3975(94)00041-G.
- 11 Chung-Kil Hur, Georg Neis, Derek Dreyer, and Viktor Vafeiadis. The power of parameterization in coinductive proof. In *POPL*, pages 193–206. ACM, 2013. doi:10.1145/2429069.2429093.
- 12 B. Knaster. Un théorème sur les fonctions d'ensembles. *Annales de la Société Polonaise de Mathématiques*, 6:133–134, 1928.
- 13 D. Kozen. A Completeness Theorem for Kleene Algebras and the Algebra of Regular Events. *Information and Computation*, 110(2):366–390, 1994. doi:10.1006/inco.1994.1037.
- 14 D. Kozen. Kleene algebra with tests. *Transactions on Programming Languages and Systems*, 19(3):427–443, May 1997. doi:10.1145/256167.256195.
- 15 Daniel Kroh. A Complete System of B-Rational Identities. In *ICALP*, volume 443 of *LNCS*, pages 60–73. Springer, 1990. doi:10.1007/BFb0032022.
- 16 Marina Lenisa, John Power, and Hiroshi Watanabe. Distributivity for endofunctors, pointed and co-pointed endofunctors, monads and comonads. *Electronical Notes in Computer Science*, 33:230–260, 2000. doi:10.1016/S1571-0661(05)80350-0.
- 17 Yoshiaki Nakamura. Partial derivatives on graphs for Kleene allegories. In *LiCS*, pages 1–12. IEEE, 2017. doi:10.1109/LICS.2017.8005132.
- 18 Damien Pous. Kleene Algebra with Tests and Coq tools for while programs. In *ITP*, volume 7998 of *LNCS*, pages 180–196. Springer, 2013. doi:10.1007/978-3-642-39634-2_15.
- 19 Damien Pous. Coinduction all the way up. In *LICS*, pages 307–316. ACM, 2016. doi:10.1145/2933575.2934564.
- 20 Damien Pous. On the positive calculus of relations with transitive closure. In *STACS*, volume 96 of *LIPICs*, pages 3:1–3:16. Schloss Dagstuhl, 2018. doi:10.4230/LIPICs.STACS.2018.3.
- 21 Damien Pous and Jurriaan Rot. Companions, Codensity, and Causality. In *FoSSaCS*, volume 10203 of *LNCS*. Springer, 2017. Extended version at <https://arxiv.org/abs/1712.08526>. doi:10.1007/978-3-662-54458-7_7.
- 22 Jan J. M. M. Rutten. A coinductive calculus of streams. *Math. Struct. in Comp. Sci.*, 15(1):93–147, 2005. doi:10.1017/S0960129504004517.

4:4 Coinduction: Automata, Formal Proof, Companions

- 23 D. Sangiorgi. On the Bisimulation Proof Method. *Math. Struct. in Comp. Sci.*, 8:447–479, 1998. doi:10.1017/S0960129598002527.
- 24 Davide Sangiorgi and David Walker. *The π -calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.
- 25 A. Tarski. A Lattice-Theoretical Fixpoint Theorem and its Applications. *Pacific Journal of Mathematics*, 5(2):285–309, June 1955.

Ω -Automata: A Coalgebraic Perspective on Regular ω -Languages

Vincenzo Ciancia

Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo" - Consiglio Nazionale delle Ricerche, Pisa, Italy

vincenzo.ciancia@isti.cnr.it

Yde Venema

Institute for Logic, Language and Computation, Universiteit van Amsterdam, Amsterdam, The Netherlands

<https://staff.fnwi.uva.nl/y.venema/>

y.venema@science.uva.nl

Abstract

In this work, we provide a simple coalgebraic characterisation of regular ω -languages based on languages of *lassos*, and prove a number of related mathematical results, framed into the theory of a new kind of automata called Ω -automata. In earlier work we introduced Ω -automata as two-sorted structures that naturally operate on *lassos*, pairs of words encoding ultimately periodic streams (infinite words). Here we extend the scope of these Ω -automata by proposing them as a new kind of acceptor for *arbitrary* streams. We prove that Ω -automata are expressively complete for the regular ω -languages. We show that, due to their coalgebraic nature, Ω -automata share some attractive properties with deterministic automata operating on finite words, properties that other types of stream automata lack. In particular, we provide a simple, coalgebraic definition of bisimilarity between Ω -automata that exactly captures language equivalence and allows for a simple minimization procedure. We also prove a coalgebraic Myhill-Nerode style theorem for lasso languages, and use this result, in combination with a closure property on stream languages called *lasso determinacy*, to give a characterization of regular ω -languages.

2012 ACM Subject Classification Theory of computation \rightarrow Formal languages and automata theory; Theory of computation \rightarrow Automata over infinite objects

Keywords and phrases ω -automata, regular ω -languages, coalgebra, streams, bisimilarity

Digital Object Identifier 10.4230/LIPIcs.CALCO.2019.5

Acknowledgements The authors wish to thank Oded Maler and Ludwig Staiger for useful discussions about right congruences and characterisations of ω -regularity.

1 Introduction

The theory of finite automata, seen as devices for classifying (possibly) infinite structures [11], combines a rich mathematical theory, dating back to the seminal work of Büchi, Rabin, and others, with a wide range of applications, in areas related to the verification and synthesis of systems that are not supposed to terminate. This applies in particular to automata operating on streams (infinite words): stream automata (or ω -automata), see [15] for a comprehensive reference. Stream automata can be classified in terms of their acceptance conditions (e.g. parity, Muller, Büchi), and come in deterministic, nondeterministic and alternating variants. With the exception of the weaker deterministic Büchi automata, these models all recognize the same class of stream languages (or ω -languages), viz., the *regular* ones.

Our perspective on stream automata and regular ω -languages will be coalgebraic. Universal Coalgebra [17] is a mathematical, category-based theory of evolving state-based systems such as streams, (infinite) trees, Kripke models, (probabilistic) transition systems, and many others. This approach combines simplicity with generality and wide applicability:



© Vincenzo Ciancia and Yde Venema;

licensed under Creative Commons License CC-BY

8th Conference on Algebra and Coalgebra in Computer Science (CALCO 2019).

Editors: Markus Roggenbach and Ana Sokolova; Article No. 5; pp. 5:1–5:18

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

many features, including input, output, nondeterminism, probability, and interaction, can easily be encoded in the coalgebra type, which is formally an endofunctor \mathbb{T} on some base category \mathbb{C} (often \mathbf{Set} , the category with sets as objects and functions as arrows). When it comes to the coalgebraic perspective on automata theory, the standard deterministic finite automata (DFAs) operating on *finite* words have been recognized as paradigmatic examples of coalgebras [16]: Standard coalgebraic concepts such as final coalgebras and bisimulations, feature very naturally in the theory of DFAs. For instance, language equivalence between DFA states is captured exactly by the notion of bisimilarity, an observation yielding both a coinductive procedure to decide language equivalence and an elegant method for minimizing automata; states of the final coalgebra also represent equivalence classes of the celebrated Myhill-Nerode theorem. These observations naturally raise the question whether automata operating on streams admit an attractive coalgebraic presentation as well.

Exactly this problem was addressed in our earlier paper [9]. Based on the well-known characterization of regular ω -languages by their ultimately periodic (UP) fragment, we focused on finite representations of UP streams called *lassos*: a lasso is a pair (u, v) of a finite word u and a finite nonempty word v , representing the UP stream uv^ω . This approach built on the work of Calbrix, Nivat & Podelski [7], who introduced certain DFAs operating on finite words of the form $u\$v$, where $\$$ is a special symbol separating the spoke u from the loop v of a lasso. The contribution of [9] was threefold. First, we introduced *two-sorted* automata operating on lassos directly, and we showed that these lasso automata share some nice properties with standard DFAs. Second, we presented these lasso automata as coalgebras, for a functor Ω on the category \mathbf{Set}^2 of two-sorted sets with two-sorted functions. And third, we identified two properties, *coherence* and *circularity*, that characterize those Ω -coalgebras of which the recognized lasso language correspond to the UP fragment of a regular ω -language.

Where our discussion in [9] stayed at a fairly abstract level, and we only considered acceptance of lassos, the current paper shows how to make concrete use of Ω -coalgebras¹ as automata operating on arbitrary streams. For this purpose, we introduce a new kind of stream automaton by defining an Ω -*automaton* as a circular and coherent lasso automaton, and endowing these structures with a suitable notion of acceptance for streams.

Contribution

The technical results that we prove on these Ω -automata include the following:

- we prove that the property of being an Ω -automaton is decidable; that is, we show that it is decidable whether a given lasso automaton is circular and coherent (Theorem 18);
- we show that, with respect to expressive power, Ω -automata exactly capture the regular ω -languages (Theorem 22 and Corollary 24);
- we show that for Ω -automata, the natural (and coalgebraic) notion of bisimilarity exactly captures language equivalence (Theorem 25), and
- as a corollary we obtain a simple and natural minimization procedure for Ω -automata (Theorem 28), which is an instance of the well-known coalgebraic partition refinement;
- we prove a Myhill-Nerode theorem for lassos (Theorem 34), which is closely related to the work of Maler & Staiger [14], but has a coalgebraic component involving the final Ω -coalgebra;
- as a corollary of this, we give a new characterization of regular ω -languages (Theorem 37).

¹ In fact, the automata that we consider in this paper are a simplification of the ones defined in [9], see Remark 3. All results proved in [9] also apply in this setting, with only trivial modifications to the proofs.

Combining these observations with the results obtained in [9], we find that Ω -automata share many of the attractive properties of DFAs: bisimulation captures language equivalence and can be used for a minimization procedure, regular ω -languages correspond to the finitely generated subcoalgebras of the final coalgebra, Boolean operations (including complementation) on regular ω -languages can be implemented by straightforward operations on Ω -coalgebras, etc. This is in sharp contrast to the setting of standard devices such as Büchi or parity automata, where to the best of our knowledge no satisfactory notion of bisimilarity between automata has been defined.

The main point of this article, then, lies not so much in the above list of individual contributions, but in the picture we obtain by putting all technical results (from this paper and from [9]) *together*. The emerging picture shows that stream automata and (regular) ω -languages *do* fit in a coherent coalgebraic framework, and one that shares many of the attractive properties of DFAs and regular languages of finite words.

Related work. In between the publication of [9] and this work, some relevant results on coalgebraic interpretations of stream automata appeared in the literature. The research line of [20, 19] follows the so-called Kleisli approach to trace semantics of coalgebras. Here a system is a coalgebra in the Kleisli category associated with some monad T that encodes the branching style exhibited by the system. In such a Kleisli category the homsets are often equipped with a natural order relation, and the authors use this fact to characterize the behaviour of parity-style automata as an arrow that is a solution of some hierarchical equation system over this order. Proving the effectiveness of their definition, the authors are able to capture not only regular ω -languages, but also various forms of so-called *ω -regular behavioural equivalences* of a labelled graph, including *infinite trace semantics*, *tree languages*, and systems with both non-deterministic and probabilistic branching. In [6] similar results are developed for systems with so-called *internal moves* and the corresponding notion of *weak bisimilarity*.

Attractive about this perspective is the modularity of the coalgebraic approach, which permits results to be generalised to various interpretations of the notion “(ω -regular) behaviour”, and its clear link to the research area of *process calculi* and the categorical modelling of their behavioural equivalences; however, such abstract representations are not directly exploitable for the verification of properties related to such equivalences. In contrast, our approach is framed in the logical-algorithmic view of automata, and therefore aimed at simple, finite representations, and algorithms such as minimization (see Section 4.4) or Boolean operations (defined in [9], including complementation) that are also a typical ingredient of model checkers. This does *not* mean that our presentation lacks generality; for instance, changing the base category is an interesting possibility to explore (consider e.g. [8], defining a class of *nominal* omega-regular languages, notably also characterised by a form of ultimately periodic behaviour, see Theorem 7 therein).

To mention some closely related work in a different direction, a well-known algorithmic application in language theory is *automata learning* [1]. It is noteworthy that, in fact, a class of finite-state machines [2], similarly inspired by [14], has been used by Angluin and Fisman for learning ω -regular languages [3], and exploited by an independent group in a tool that obtained best-in-class results in terms of computational efficiency [12]. We believe that the coalgebraic perspective brings in some unique improvements on its own; for instance, minimization and Myhill-Nerode style theorems are a standard consequence of the theory; furthermore, the theory of coalgebras opens up to the possibility of generalising the presented constructions, and derive new ones (more on this in the Conclusions).

2 Preliminaries

Assuming that the reader is familiar with the theory of automata operating on (in)finite words [15], we fix some notation and terminology.

Given sets X, Y and Z , we define Y^X as the function space of maps from X to Y , and using currying we may identify the sets $Z^{X \times Y}$ and $(Z^Y)^X$.

Throughout this paper we fix a finite alphabet C consisting of symbols or colors. We write C^* (C^+) for the set of finite (respectively, finite nonempty) *words* over C . The empty word is denoted as ϵ , and the length of a word u as $|u|$. With ω denoting the set of natural numbers, C^ω is the collection of *streams* (*infinite words*) over C . The binary operation of concatenation between finite and (in)finite words is denoted by juxtaposition. For $u \in C^+$, we let u^ω denote the stream that repeats u ω many times, and for $(v_i)_{i \in \omega}$ in C^+ we write \vec{v} for the stream $v_0 v_1 \dots$. A stream α is *ultimately periodic* if it is of the form uv^ω for some $u \in C^*$ and $v \in C^+$. A *language* is a set of finite words; an ω -*language* or *stream language* is a set of streams. The *ultimately periodic fragment* $UP(L)$ of a stream language L is the set of its ultimately periodic members.

A *transition function* on a set X is a map of the form $\rho : X \times C \rightarrow X$ (or, equivalently, $\rho : X \rightarrow X^C$). Given such a map, we inductively define the functions $\hat{\rho} : X \times C^* \rightarrow X$ and $\rho^\circ : X \times C^* \rightarrow \mathcal{P}X$ by putting $\hat{\rho}(x, \epsilon) := x$, $\hat{\rho}(x, cu) := \hat{\rho}(\rho(x, c), u)$, resp. $\rho^\circ(x, \epsilon) := \emptyset$, $\rho^\circ(x, cu) := \{\rho(x, c)\} \cup \rho^\circ(\rho(x, c), u)$. In words, $\hat{\rho}(p, u)$ denotes the state reached by a transition system after, starting at state p , it has processed the word u ; and $\rho^\circ(p, u)$ is the set of states passed along the way, after leaving p . We often write $x \xrightarrow{c}_\rho y$ for $\rho(x, c) = y$ and $x \xrightarrow{u}_\rho y$ for $\hat{\rho}(x, u) = y$, omitting the subscript if ρ is understood.

Our concept of an automaton will not include an initial state; rather, we define an *initialized* or *pointed* automaton to be a pair (\mathbb{A}, a) such that a is a state of the automaton \mathbb{A} . Given a transition map $\rho : X \times C \rightarrow X$, a state $x \in X$, and a stream α , we let $Inf(x, \alpha)$ denote the set of states in X traversed infinitely often when following α starting from x . A (deterministic) *parity* automaton is a triple $\mathbb{P} = (P, \rho, \Pi)$ such that P is a finite set of states, $\rho : P \times C \rightarrow P$ is a transition map, and $\Pi : P \rightarrow \omega$ is a priority function. An initialized parity automaton (\mathbb{P}, p) accepts a stream α if $\max(\Pi[Inf(p, \alpha)])$ is even. The sets of words/nonempty words/streams recognized by an initialized automaton (\mathbb{A}, a) (of the appropriate kind) are denoted as $L(\mathbb{A}, a)/L^+(\mathbb{A}, a)/Streams(\mathbb{A}, a)$. We shall generally use the symbol \triangleright for acceptance. A stream language is *regular* if it is recognized by some initialized parity automaton.

3 Ω -coalgebras as lasso automata

3.1 Basics

As mentioned in the introduction, regular stream languages are determined by their ultimately periodic fragments. This motivates our interest in finite representations of ultimately periodic streams: lassos.

► **Definition 1.** A lasso is a pair $(u, v) \in C^* \times C^+$, representing the stream uv^ω . The words u and v are respectively the spoke and loop of the lasso. We call two lassos (u_0, v_0) and (u_1, v_1) bisimilar, notation: $(u_0, v_0) \stackrel{\sim}{\Leftarrow} (u_1, v_1)$, if they represent the same stream, i.e., if $u_0 v_0^\omega = u_1 v_1^\omega$.

Continuing the program of Calbrix, Nivat & Podelski [7] in a coalgebraic direction, in [9] we introduced Ω -coalgebras as automata operating on lassos.

► **Definition 2.** An Ω -coalgebra is a structure $\mathbb{A} = (P, X, \rho, \xi, \sigma, F)$ such that P and X are sets of spoke and loop states, respectively; both $\rho : P \rightarrow P^C$ and $\xi : X \rightarrow X^C$ are transition functions; $\sigma : P \rightarrow X^C$ is the switch map; and $F \subseteq X$ is a set of accepting states. A lasso automaton is a finite Ω -coalgebra.

The loop automaton of \mathbb{A} is the DFA $\mathbb{A}_\ell := (P \uplus X, \sigma; \xi, F)$, where $\sigma; \xi : (P \uplus X) \rightarrow (P \uplus X)^C$ is defined as σ on P and as ξ on X . For $p \in P$, we define $\text{Loop}(p)$ as the set of finite words accepted by (\mathbb{A}_ℓ, p) .

Think of $\mathbb{A} = (P, X, \rho, \xi, \sigma, F)$ as a spoke part (P, ρ) and a loop part (X, ξ, F) that are connected by the switch function $\sigma : P \times C \rightarrow X$. Clearly this “loop part” is a DFA in its own right, just like “the” loop automaton \mathbb{A}_ℓ ; observe that (X, ξ, F) is a subautomaton of the loop automaton \mathbb{A}_ℓ , and that by construction, any initialized automaton (\mathbb{A}_ℓ, p) with $p \in P$ will accept nonempty words only.

► **Remark 3.** In fact, Definition 2 is a simplification of the one given in [9], where the switching function has type $\sigma : P \rightarrow X$, and the loop part is a finite automaton designed to operate on nonempty words.

► **Remark 4.** Although the main point of the paper is to show how stream automata nicely fit a coalgebraic framework and our entire approach is coalgebraic in spirit, we decided to keep the categorical machinery at a minimum. The coalgebraic definition is phrased as follows. Our base category is Set^2 , the category of two-sorted sets with two-sorted functions. For the definition of the endofunctor $\Omega : \text{Set}^2 \rightarrow \text{Set}^2$, it will be convenient to use the functors $\text{E}_C := (-)^C$ and $\text{D}_C := 2 \times (-)^C$ of, respectively, (C) -transition functions and (C) -DFAs. Now, given an object (X_0, X_1) in Set^2 , we define

$$\Omega(X_0, X_1) := (\text{E}_C X_0 \times \text{E}_C X_1, \text{D}_C X_1).$$

For the action of Ω on arrows, consider a pair $f = (f_0, f_1)$ of functions $f_i : X_i \rightarrow Y_i$, then Ωf is the pair of functions $(\text{E}_C f_0 \times \text{E}_C f_1, \text{D}_C f_1)$. With this definition, the Ω -coalgebras of Definition 2 and 6 are coalgebras for the functor Ω indeed; to see this, observe that a coalgebra for the functor Ω consists of two sets X_0 and X_1 , and three maps $h : X_0 \rightarrow X_0^C$, $h' : X_0 \rightarrow X_1^C$, and $h'' : X_1 \rightarrow 2 \times X_1^C$. An Ω -coalgebra $(P, X, \rho, \xi, \sigma, F)$ is rendered in such a form by letting $X_0 = P$, $X_1 = X$, $h = \rho$, $h' = \xi$, and deriving h'' from σ and F . As the construction is similar to that of [9], we leave the details as an exercise for the reader.

► **Definition 5.** Where $\mathbb{A} = (P, X, \rho, \xi, \sigma, F)$ is an Ω -coalgebra, and $p \in P$ is a spoke state, we say that a lasso (u, v) is accepted by \mathbb{A} at p , notation: $\mathbb{A}, p \triangleright (u, v)$, if $\widehat{\sigma; \xi}(\widehat{\rho}(p, u), v) \in F$, and we write $\text{Lassos}(\mathbb{A}, p)$ to denote the lasso language recognized by (\mathbb{A}, p) , that is, the set of all lassos accepted by $\text{Lassos}(\mathbb{A}, p)$. Finally, a lasso language is called regular if it is the language recognized by a pointed finite Ω -coalgebra, i.e., a lasso automaton.

Intuitively, an Ω -coalgebra operates on a lasso (u, v) by first processing the spoke u , then switching to the loop automaton and processing the loop v . It accepts the lasso iff the resulting state is accepting. That is, (\mathbb{A}, p) accepts (u, v) iff $v \in \text{Loop}(\widehat{\rho}(p, u))$.

► **Definition 6.** Let $\mathbb{A} = (P, X, \rho, \xi, \sigma, F)$ and $\mathbb{A}' = (P', X', \rho', \xi', \sigma', F')$ be two Ω -coalgebras. An Ω -morphism from \mathbb{A} to \mathbb{A}' is a pair $h = (h_0, h_1)$ of maps $h_0 : P \rightarrow P'$, $h_1 : X \rightarrow X'$ such that, for all $p \in P, x \in X$ and $c \in C$ we have

$$\text{(M1)} \quad h_0(\rho(p, c)) = \rho'(h_0 p, c),$$

$$\text{(M2)} \quad h_1(\xi(x, c)) = \xi'(h_1 x, c),$$

$$\text{(M3)} \quad h_1(\sigma(p, c)) = \sigma'(h_0 p, c), \text{ and}$$

(M4) $x \in F$ iff $h_1x \in F'$.

We usually write h for h_i .

A spoke state p of \mathbb{A} generates a subcoalgebra \mathbb{A}_p of \mathbb{A} that is based on the sets of spoke and loop states that are reachable from p (in the obvious sense, using ρ, σ , and ξ).

As a manifestation of the coalgebraic nature of our structures, we discuss below how the natural concept of equivalence induced by Ω -morphisms can be captured by a notion of bisimilarity.

► **Definition 7.** A bisimulation between two Ω -coalgebras \mathbb{A} and \mathbb{A}' is a pair $Z = (Z_0, Z_1)$ of relations $Z_0 \subseteq P \times P'$, $Z_1 \subseteq X \times X'$ such that, for all $(p, p') \in Z_0$ and $(x, x') \in Z_1$, and all $c \in C$ we have

(B1) $(\rho(p, c), \rho'(p', c)) \in Z_0$,

(B2) $(\xi(x, c), \xi'(x', c)) \in Z_1$,

(B3) $(\sigma(p, c), \sigma'(p', c)) \in Z_1$, and

(B4) $x \in F$ iff $x' \in F'$.

Two pointed coalgebras (\mathbb{A}, p) and (\mathbb{A}', p') are bisimilar, notation: $\mathbb{A}, p \Leftrightarrow \mathbb{A}', p'$, if there is a bisimulation linking p and p' .

The following characterization of bisimilarity using morphisms holds for a wide range of coalgebras; in coalgebraic terms, it says that for the functor Ω , the notions of bisimilarity and behavioral equivalence coincide. The proposition follows from categorical properties of the functor Ω , but also has a straightforward direct proof.

► **Proposition 8.** Let (\mathbb{A}, p) and (\mathbb{A}', p') be pointed Ω -coalgebras. Then $(\mathbb{A}, p) \Leftrightarrow (\mathbb{A}', p')$ iff there is a Ω -coalgebra \mathbb{B} and Ω -morphisms $h : \mathbb{A} \rightarrow \mathbb{B}$ and $h' : \mathbb{A}' \rightarrow \mathbb{B}$ such that $hp = h'p'$.

The following proposition on bisimilarity will be needed later on; we omit the proof which follows a routine coalgebra argument.

► **Proposition 9.** Let \mathbb{A} and \mathbb{A}' be two Ω -coalgebras. Then

(1) the collection of bisimulations between \mathbb{A} and \mathbb{A}' forms a complete lattice, of which the join is given by union;

(2) the relation \Leftrightarrow itself is the largest bisimulation between \mathbb{A} and \mathbb{A}' ;

(3) if $\mathbb{A} = \mathbb{A}'$, the relation \Leftrightarrow is an equivalence relation.

The key observation is that bisimilarity *exactly* captures lasso equivalence. This was first shown in [9]; by looking at the explicit definition of bisimilarity given in Definition 7, the proof is a simple extension to the two-sorted setting of the classical result that in classical DFAs operating on *finite* words, bisimilarity coincides with language equivalence (see [16]).

► **Fact 10.** [9] Any pair of pointed Ω -coalgebras (\mathbb{A}, p) and (\mathbb{A}', p') satisfy

$$\mathbb{A}, p \Leftrightarrow \mathbb{A}', p' \text{ iff } \text{Lassos}(\mathbb{A}, p) = \text{Lassos}(\mathbb{A}', p'). \quad (1)$$

► **Example 11.** Fixing the alphabet $C = \{a, b\}$, we define the Ω -coalgebra $\mathbb{A} = (P, X, \rho, \xi, \sigma, F)$ where $P = \{1, 2, 3\}$, $X = \{4, 5, 6\}$, $\rho = \{(1, a) \mapsto 1, (1, b) \mapsto 2, (2, a) \mapsto 1, (2, b) \mapsto 3, (3, a) \mapsto 2, (3, b) \mapsto 3\}$, $\xi = \{(4, a) \mapsto 4, (4, b) \mapsto 6, (5, b) \mapsto 5, (5, a) \mapsto 6, (6, a) \mapsto 6, (6, b) \mapsto 6\}$, $\sigma = \{(p, a) \mapsto 4, (p, b) \mapsto 5\}$ for each $p \in P$, and $F = \{4, 5\}$. By construction, for all $p \in P$, we have $\text{Lassos}(\mathbb{A}, p) = \{(u, v) \mid u \in C^* \wedge v \in (\{a^+\} \cup \{b^+\})\}$. Consider the Ω -coalgebra $\mathbb{A}' = (\{1\}, X, \rho', \sigma', \xi, F)$, where ρ' and σ' are the restriction of ρ and σ , respectively, to the singleton $\{1\}$. By Fact 10, $\mathbb{A}, 1 \Leftrightarrow \mathbb{A}, 2 \Leftrightarrow \mathbb{A}, 3$. The situation is witnessed by the Ω -morphism $h : \mathbb{A} \rightarrow \mathbb{A}'$, which identifies all the states in P by mapping them to the state 1. Furthermore, for all $p \in P$, we have $\mathbb{A}, p \Leftrightarrow \mathbb{A}', 1$.

3.2 From parity automata to lasso automata

Given a Büchi automaton \mathbb{B} , Calbrix, Nivat & Podelski [7] constructed a DFA recognizing the finite-word language $\{u\$v \mid \mathbb{B} \triangleright uv^\omega\}$, where $\$$ is a new symbol (i.e., not in C). Here we give a similar construction for parity automata.

► **Definition 12.** Let $\mathbb{P} = (P, \rho, \Pi)$ be a parity automaton and let p be a state of \mathbb{P} . The DFA $\mathbb{X}_p := (X, \xi, F_p)$ is based on the state space $X := (P \times N)^P$, where $N := \text{Ran}(\Pi)$ is the range of Π . To define its transition map ξ , consider an arbitrary state $t \in X$ and think of t as the pairing of $t^0 : P \rightarrow P$ and $t^1 : P \rightarrow N$. Now define

$$\xi(t)(c) := \lambda q. \left(\rho(t^0 q, c), \max(t^1 q, \Pi(\rho(t^0 q, c))) \right)$$

For the set F_p of accepting states, define, for an arbitrary state $t \in X$, the sequence $(p_i^t)_{i \in \omega}$ by putting $p_0^t := p$, $p_{i+1}^t := t^0 p_i^t$, and put

$$F_p := \{t \in X \mid \max(t^1[\text{Inf}((p_i^t)_{i \in \omega})]) \text{ is even}\},$$

where $\text{Inf}((p_i^t)_{i \in \omega})$ is the set of $p \in P$ occurring as p_i^t for infinitely many i . Finally, we define $s_I^p \in X$ as the initial state $s_I^p := \lambda q.(q, 0)$.

Intuitively, the initialized DFA (\mathbb{X}_p, s_I^p) consumes a word v by following it in parallel, starting from each state of \mathbb{P} . Moreover, in each of these parallel runs the automaton collects the maximum priority of the traversed states. This explains the carrier and the transition map of the automaton \mathbb{X}_p . For its set of accepting states, first note that F_p is the only part of \mathbb{X}_p depending on p . The idea behind its definition is that for a word $v \in C^+$, the state $t := \widehat{\xi}(s_I^p, v)$ encodes essential information on the run (\mathbb{P}, p) on the stream v^ω . In particular, we have $p_i^t = \widehat{\rho}(p, v^i)$, for all i . Analyzing the way in which the map $t^1 : P \rightarrow N$ keeps track of maximal priorities along finite runs, we may then show that $\max(t^1[\text{Inf}((p_i^t)_{i \in \omega})])$ corresponds to the maximal priority that one encounters in the run of (\mathbb{P}, p) on v^ω .

The key observation on this automaton is that (\mathbb{X}_p, s_I^p) recognizes the looping language of (\mathbb{P}, p) , that is, for all $v \in C^+$:

$$(\mathbb{X}_p, s_I^p) \text{ accepts } v \text{ iff } (\mathbb{P}, p) \text{ accepts } v^\omega.$$

► **Definition 13.** Let $\mathbb{P} = (P, \rho, \Pi)$ be a parity automaton. Recalling that by definition, P is finite, we define the lasso automaton $\mathbb{A}_\mathbb{P} := (P, X, \rho, \xi, \sigma, F)$ by letting (X, ξ, F) be the coproduct (disjoint union) of the family $\{\mathbb{X}_p \mid p \in P\}$ of DFAs, and putting $\sigma(p, c) := \xi(s_I^p, c)$.

► **Fact 14.** [9] Let (\mathbb{P}, p) be an initialized parity automaton. For any lasso $(u, v) \in C^* \times C^+$:

$$(\mathbb{A}_\mathbb{P}, p) \text{ accepts } (u, v) \text{ iff } (\mathbb{P}, p) \text{ accepts } uv^\omega. \quad (2)$$

► **Example 15.** Using the alphabet $C = \{a, b\}$, consider the parity automaton $\mathbb{P} = (P, \rho, \Pi)$ where P and ρ come from Example 11, and $\Pi = \{1 \mapsto 0, 3 \mapsto 0, 2 \mapsto 1\}$. It is easily seen that, no matter what the initial state is, the language accepted by the automaton is $\{\alpha \subseteq C^\omega \mid \exists i \in \omega. \exists c \in C. \forall j \geq i. \alpha_j = c\}$, that is, those streams that have a tail consisting of an infinite repetition of one symbol. The reader should note that there is no single-state parity automaton accepting the same language, as a single-state automaton may either accept C^ω or the empty language. However, either by direct construction, or by Fact 14, for all $p \in P$, it can be shown that $\mathbb{A}_\mathbb{P}, p \Leftrightarrow \mathbb{A}', 1$, where \mathbb{A}' comes from Example 11, in turn.

3.3 Coherence & Circularity

The language recognized by a lasso automaton (\mathbb{A}, p) does not necessarily correspond to the ultimately periodic fragment of a regular ω -language. A necessary condition for the latter is that $\text{Lassos}(\mathbb{A}, p)$ is *invariant under lasso bisimilarity*: $(u, v) \Leftrightarrow (u', v')$ implies that $(u, v) \in \text{Lassos}(\mathbb{A}, p)$ iff $(u', v') \in \text{Lassos}(\mathbb{A}, p)$. In [9] we proved that this condition is also sufficient, and we characterized it by the properties of coherence and circularity.

► **Definition 16.** *A regular language L is circular if $v \in L \Leftrightarrow v^k \in L$, for all $k > 0$ and $v \in C^+$. An initialized DFA (\mathbb{A}, a) is circular if $L(\mathbb{A}, a)$ is circular. A lasso automaton \mathbb{A} is circular if each $\text{Loop}(p)$ is circular, and coherent if $cu \in \text{Loop}(p) \Leftrightarrow uc \in \text{Loop}(\rho(p, c))$, for every spoke state p , $u \in C^*$ and $c \in C$.*

► **Fact 17.** [9] *For any lasso automaton $\mathbb{A} = (P, X, \rho, \xi, \sigma, F)$ the following are equivalent:*

- (1) $\forall p \in P$. $\text{Lassos}(\mathbb{A}, p) = \{(u, v) \mid uv^\omega \in L\}$ for some regular ω -language L ;
- (2) $\forall p \in P$. $\text{Lassos}(\mathbb{A}, p)$ is bisimulation invariant;
- (3) \mathbb{A} is circular and coherent.

Motivated by Fact 17, in the sequel we will largely confine our attention to circular and coherent lasso automata. This explains the importance of the following result.

► **Theorem 18.** *It is decidable whether a given lasso automaton is circular and coherent.*

Finally, we note that by Fact 10, the class of circular and coherent lasso automata is closed under taking surjective Ω -morphisms.

4 Ω -automata

In this section, we look at Ω -automata as acceptors of *streams*. As we shall see, this notion of acceptance coincides with the one of parity automata (Theorem 22) and is invariant under Ω -morphisms (Theorem 23). The main goal of this section is to show that, unlike the standard types of stream automata, Ω -automata admit a natural notion of bisimilarity that *exactly* captures language equivalence (Theorem 25). Finally, as a corollary of these results we obtain a simple and natural minimization procedure for Ω -automata (Theorem 28).

4.1 Ω -coalgebras as stream automata

In the previous section we saw that a lasso language corresponds to the ultimately periodic fragment of a regular ω -language if and only if it is the language recognized by an initialized, circular and coherent lasso automaton. This suggests that circular and coherent Ω -coalgebras might be used directly as stream automata, and inspires the following definition.

► **Definition 19.** *An Ω -automaton is a circular and coherent lasso automaton.*

Following ideas in Calbrix, Nivat & Podelski [7], we now define acceptance of streams.

► **Definition 20.** *Let $\mathbb{A} = (P, X, \rho, \xi, \sigma, F)$ be an Ω -automaton, and let q be a spoke state of \mathbb{A} . We say that a stream α is accepted by (\mathbb{A}, q) , notation: $(\mathbb{A}, q) \triangleright \alpha$, if there are a finite word u , a sequence $(v_i)_{i \in \omega}$ of finite, non-empty words, a state $p \in P$ and an accepting state $z \in F$ such that $\alpha = uv^\omega$, $q \xrightarrow{u}_\rho p$ and $p \xrightarrow{v_i}_\rho p$, $p \xrightarrow{v_i}_{\sigma \cdot \xi} z$ for each $i \in \omega$. The set of streams accepted by an initialized Ω -automaton (\mathbb{A}, p) is denoted as $\text{Streams}(\mathbb{A}, p)$.*

► **Remark 21.** Where this notion of acceptance may seem somewhat odd at first sight, it can be related to that of well-known stream automata. A *successful run* of (\mathbb{A}, q) on a stream α consists of

- a run of the spoke part of \mathbb{A} on a (finite) initial segment u of α (i.e., $\alpha = u\beta$ for some stream β), leading to a spoke state p , followed by
- an infinite run of the *product structure* of (P, ρ) and \mathbb{A}_ℓ on the remaining stream β , where for some accepting state $z \in F$, the product automaton infinitely often makes a *silent step* from (p, z) to (p, p) :

$$(p, p) \xRightarrow{\rho \times (\sigma; \xi)} (p, z) \xrightarrow{\epsilon} (p, p) \xRightarrow{\rho \times (\sigma; \xi)} (p, z) \xrightarrow{\epsilon} (p, p) \xRightarrow{\rho \times (\sigma; \xi)} \dots$$

Based on this observation, it is not difficult to show (but rather tedious to spell out in detail) that Ω -automata can be seen as rather special Büchi automata, where the nondeterminism is restricted to (1) a unique jump from an initial part of the automaton to a final part, and (2) some very specific silent steps. From this perspective, that is, with Ω -automata taken as a subclass of Büchi automata with silent steps, it is remarkable that the theory of Ω -automata is so well-behaved when we consider bisimilarity etc. This good behavior may be explained by the observation that seen as lasso automata, Ω -coalgebra are completely deterministic.

4.2 Adequacy

The following theorem states that, when it comes to recognizing stream languages, Ω -automata are as least as expressive as more standard models like parity automata.

► **Theorem 22 (Adequacy).** *Let (\mathbb{P}, q) be an initialized deterministic parity automaton. Then $\text{Streams}(\mathbb{P}, q) = \text{Streams}(\mathbb{A}_{\mathbb{P}}, q)$.*

4.3 Language equivalence as bisimilarity

► **Theorem 23 (Invariance).** *Let $h : \mathbb{A} \rightarrow \mathbb{A}'$ be an Ω -morphism between two Ω -automata.*

$$\text{Streams}(\mathbb{A}, q) = \text{Streams}(\mathbb{A}', hq) \tag{3}$$

for any spoke state q of \mathbb{A} .

► **Corollary 24.** *Let (\mathbb{A}, q) be an Ω -automaton. Then $\text{Streams}(\mathbb{A}, q)$ is an ω -regular language and $\text{Lassos}(\mathbb{A}, q) = \{(u, v) \mid uv^\omega \in \text{Streams}(\mathbb{A}, q)\}$.*

The next result shows that, unlike well-known types of stream automata such as Büchi, Muller or parity automata, Ω -automata share a fundamental property with deterministic automata operating on finite words.

► **Theorem 25 (Language equivalence as bisimilarity).** *Let (\mathbb{A}, q) and (\mathbb{A}', q') be two initialized Ω -automata. Then*

$$\mathbb{A}, q \Leftrightarrow \mathbb{A}', q' \text{ iff } \text{Streams}(\mathbb{A}, q) = \text{Streams}(\mathbb{A}', q'). \tag{4}$$

► **Example 26.** Continuing from Example 15, observe that the Ω -coalgebras \mathbb{A} and \mathbb{A}' are actually circular and coherent, and therefore Ω -automata. The stream language that these coalgebras accept, from any initial state, is the same as the one accepted by the parity automaton \mathbb{P} , from any initial state. However, in the realm of Ω -automata, by virtue of the associated notion of bisimilarity, there is a canonical representative for the class of all pointed Ω -automata accepting the stream language of \mathbb{P} (from any state, as they all accept

the same language). It should not be difficult to guess that the canonical representative is $(\mathbb{A}', 1)$, up-to isomorphism. A formal proof needs just to show that the three states in X accept different languages (of finite words). In Section 4.4 we shall discuss computation of such a representative by partition refinement.

4.4 Minimal Ω -automaton

The minimization problem for stream automata is much harder than that for deterministic finite automata operating on finite words. In particular, regular ω -languages generally do not have a *unique* minimal automaton [18], and to the best of our knowledge, nice minimization procedures are only available for restricted classes of automata [18, 13].

This is different in the setting of Ω -automata. As a corollary of Theorem 25 we obtain a simple and natural minimization procedure for Ω -automata, linking the final coalgebra to the minimal automaton: Theorem 28. A *partition refinement* algorithm is a standard consequence of the coalgebraic framework (see [9] for a more detailed explanation).

► **Definition 27.** Let $\mathbb{A} = (P, X, \rho, \xi, \sigma, F)$ be an Ω -automaton, and recall from Proposition 8 that \leftrightarrow is an equivalence relation on P and on X . We denote the equivalence class of a state a with \bar{a} . Since \leftrightarrow is itself a bisimulation relation, the following is a correct definition of an Ω -coalgebra structure on the \leftrightarrow -cells:

$$\begin{aligned} \bar{\rho}(\bar{p}) &:= \overline{\rho p}, \\ \bar{\xi}(\bar{x}) &:= \overline{\xi x}, \\ \bar{\sigma}(\bar{p})(c) &:= \overline{\sigma(p)(c)}, \quad \text{for all } c, \\ \bar{F} &:= \{\bar{x} \mid x \in F\}. \end{aligned}$$

We denote the resulting Ω -automaton by $\mathbb{A}_{/\leftrightarrow}$.

The following theorem shows that $\mathbb{A}_{/\leftrightarrow}$ is a *minimal* automaton recognizing the languages of \mathbb{A} .

► **Theorem 28.** Let (\mathbb{A}, p) be an initialized Ω -automaton, with $L := \text{Streams}(\mathbb{A}, p)$. Then

- (1) $\text{Streams}(\mathbb{A}_{/\leftrightarrow}, \bar{p}) = L$;
- (2) For any initialized Ω -automaton (\mathbb{A}', p') such that $\text{Streams}(\mathbb{A}', p') = L$, there is an Ω -morphism $h : \mathbb{A}'_{p'} \rightarrow \mathbb{A}_{/\leftrightarrow}$ such that $h(p') = \bar{p}$.

In passing we note that the Theorems 22 and 28 yield a minimization procedure for parity automata (and other standard stream automata) as well; this procedure transforms any parity automaton, not into a minimal parity automaton, but into a canonically obtained minimal Ω -automaton.

► **Example 29.** Continuing from Example 26, the Ω -coalgebra \mathbb{A}' is, up to isomorphism, the minimal representative of the coalgebra \mathbb{A} . Note that the three states $\{1, 2, 3\}$ are quotiented by the unique morphism h (from Example 11).

5 A final Ω -coalgebra and a Myhill-Nerode Theorem

5.1 Final Ω -coalgebra

In the theory of Universal Coalgebra, an important role is played by final coalgebras. Recall that an object z is final in a category \mathbf{C} if for every object there is a unique arrow to z . Final coalgebras do not exist for every functor, but when they exist, they usually encode a

natural notion of *behavior* related to the functor. For instance, in the theory of DFAs, a final coalgebra is provided by the ‘language automaton’, in which the states are languages (of finite words), the transition structure is given by the derivatives ($L_c := \{u \in C^* \mid cu \in L\}$), and a language/state is accepting iff it contains the empty word. This language coalgebra has many nice properties and can be used to prove many fundamental properties of regular languages [16]. In this section we will see that the category of Ω -coalgebras admits a final coalgebra, and we will use this coalgebra to give a Myhill-Nerode theorem for regular lasso languages, and a related characterization for regular ω -languages.

► **Definition 30.** *With defining*

$$\begin{aligned} Z_0 &:= \mathcal{P}(C^* \times C^+), \\ Z_1 &:= \mathcal{P}C^*, \\ \zeta_0(L, c) &:= \{(u, v) \in C^* \times C^+ \mid (cu, v) \in L\}, \\ \zeta_1(M, c) &:= \{v \in C^* \mid cv \in M\}, \\ \sigma(L, c) &:= \{v \in C^* \mid (\epsilon, cv) \in L\}, \\ F &:= \{M \in \mathcal{P}C^* \mid \epsilon \in M\}, \end{aligned}$$

we obtain the Ω -coalgebra $\mathbb{Z} := (Z_0, Z_1, \zeta_0, \zeta_1, \sigma, F)$.

Observe that the loop part of this Ω -coalgebra is given by the final coalgebra for DFAs that we just mentioned; in particular, its carrier is based on the set of all languages of finite words. The carrier of the spoke part is given by the set $\mathcal{P}(C^* \times C^+)$ of all lasso languages. The exact relation of this Ω -coalgebra with the set of all stream languages remains to be investigated in more detail, but note that the relation of *lasso determinacy* (Definition 35) will play an important role here.

A very useful property of the structure \mathbb{Z} is that any lasso language L coincides with the set of lassos that it accepts, seen as a state of \mathbb{Z} .

► **Theorem 31.** *Let L be a lasso language. Then for any lasso (u, v) we have*

$$\mathbb{Z}, L \triangleright (u, v) \text{ iff } (u, v) \in L. \quad (5)$$

As a corollary, the relation \trianglelefteq restricts to the identity relation on \mathbb{Z} .

► **Theorem 32.** *\mathbb{Z} is final in the category of Ω -coalgebras and Ω -morphisms. That is, for every Ω -coalgebra \mathbb{A} there is a unique Ω -morphism $h : \mathbb{A} \rightarrow \mathbb{Z}$.*

5.2 A Myhill-Nerode Theorem for lasso languages

Rutten provided a nice coalgebraic perspective on the Myhill-Nerode theorem for regular languages of finite words, identifying the congruence classes of the Myhill-Nerode equivalence relation with states in the final coalgebra [16]. A similar result holds for lasso languages.

► **Definition 33.** *Let L be a lasso language. Define the equivalence relation \equiv_0 on C^* such that $u_0 \equiv_0 u_1$ iff for all lassos (u, v) it holds that $(u_0u, v) \in L \iff (u_1u, v) \in L$. Define a family of binary relations $\equiv_{[u]}$ on C^+ , indexed by the set of \equiv_0 -cells, such that $v_0 \equiv_{[u]} v_1$ iff for all $w \in C^*$, $u_0, u_1 \in [u]$ we have $(u_0, v_0w) \in L \iff (u_1, v_1w) \in L$. Finally, let*

$$(u_0, v_0) \equiv_L (u_1, v_1) \text{ iff } u_0 \equiv_0 u_1 \text{ and } v_0 \equiv_{[u_i]} v_1$$

define a relation \equiv_L on lassos.

It is obvious that \equiv_L is an equivalence relation, and here we have arrived at our coalgebraic Myhill-Nerode theorem for lassos. It refers to the generated subcoalgebra of a state/language L in the final coalgebra, see Definition 6. Recall that a lasso language is regular if it is the set of all lassos that are accepted by a pointed lasso automaton.

► **Theorem 34.** *The following are equivalent, for any lasso language L :*

- (1) L is regular;
- (2) L generates a finite subcoalgebra in \mathbb{Z} ;
- (3) the relation \equiv_L has finite index.

5.3 A characterization theorem for stream languages

Attempts at finding congruences that characterise ω -regularity date back to the earliest works in the theory of ω -languages, such as Arnold [4], who isolates the *syntactic congruence* of a regular ω -regular language $L \subseteq C^\omega$ as the coarsest congruence on C^* that recognizes L (in some precisely defined manner). An interesting open question was to identify a congruence which is of finite index if and only if a given stream language is regular. Maler and Staiger [14] approached this problem via so-called *families of right congruences* (FORCs), and proved that a stream language L is ω -regular if and only if there exists a finite FORC that recognises it. Furthermore, the paper identified a necessary and sufficient characterisation of those regular ω -languages that are accepted by a minimal state automaton, derived from Arnold's syntactic congruence.

While moving in a similar direction, we are able to simplify the matter somewhat. The definition of our lasso relation \equiv_L is reminiscent to that of a FORC, as it is dependent on equivalence classes of a right congruence on finite words.² Using the relation \equiv_L , we were able to provide an *exact* characterisation of regular ω -languages. Our characterization, Theorem 37, involves a property, *lasso determinacy*, that is somewhat related to (but not the same as) Arnold's saturation property. One may also look at lasso determinacy as an infinitary version of the *pumping* property of regular languages of finite words.

► **Definition 35.** *A stream language L is lasso determined, or has the property LD, if for every infinite sequence $(v_i)_{i \in \omega}$ of nonempty words there is an infinite set $Y \subseteq \omega$ such that*

$$\vec{v} \in L \iff (v_0 \cdots v_j)(v_{j+1} \cdots v_k)^\omega \in L.$$

for all $j, k \in Y$ with $j < k$.

It is not difficult to verify that all regular ω -languages are lasso determined. The following property justifies the terminology.

► **Proposition 36.** *Let L, L' be two stream languages with the property LD. If $\text{Lassos}(L) = \text{Lassos}(L')$ then $L = L'$.*

► **Theorem 37.** *The following are equivalent, for any stream language L :*

- (1) L is regular;
- (2) L is lasso determined and $\text{Lassos}(L)$ generates a finite subcoalgebra in \mathbb{Z} ;
- (3) L is lasso determined and the relation \equiv_L has finite index.

² However, \equiv_L does not exactly correspond to a FORC as it lacks the condition $v_0 \equiv_{[u_i]} v_1 \implies u_0 v_0 \equiv_0 u_1 v_1$.

6 Conclusions

The main point of this paper was to argue that (a slight variation of) the two-sorted lasso automaton we introduced in [9] provides an interesting framework for recognizing regular stream languages as well. For this purpose, we presented Ω -automata as the class of finite, circular and coherent lasso automata, and we defined a notion of acceptance for streams. Throughout the paper we used the fact that these structures are coalgebras for an endofunctor Ω on the category Set^2 of two-sorted sets and functions. The advantage of this coalgebraic presentation of stream automata is that bisimilarity and minimization are easily obtained using the general theory of Universal Coalgebra. Using another standard feature of the coalgebraic framework, namely, final coalgebras, we proved a Myhill-Nerode theorem for lasso automata that extends the basic framework of Rutten [16] to lassos. Then, involving a property we called *lasso determinacy*, we obtained a characterization of regular ω -languages. These results provide additional motivation for and in some sense complete the work of Maler & Staiger [14] on two-sorted congruences.

Of the many interesting directions to take from here we mention a few. First, not only Ω -automata but in particular the objects they operate on (lassos, streams) admit a very simple and natural coalgebraic presentation. It would be interesting to explore and exploit this connection further – in particular, we are interested in truly coalgebraic characterizations of regular ω -languages. In a subsequent publication we hope to report on such a characterization, which reveals the coalgebraic nature of the property of lasso determinacy and involves a pumping property for lasso languages. Second, a very interesting and useful coalgebraic concept is that of *coinduction*. This definition and proof principle has many applications in the theory of DFAs. It would be worthwhile to find and study manifestations of coinduction in the world of stream automata as well, also patterned after [20, 19, 6]. Third, given the two-sorted nature of our framework, it seems natural to explore its connections with the theory of *Wilke algebras* [21]. Finally, recent work on *coalgebraic automata learning* [5] could shed further light on the link between our framework and the research line on learning of omega-regular languages; a first step in this direction would be a mathematically precise comparison between the automata model presented in [2] and our coalgebraic variant.

Using Category Theory for modelling abstract notions entails the possibility to generalise results by changing the base category that is used. In this respect, our paper can be used as a starting point for further development of *nominal* omega-regular languages [8]. Finally, it would be interesting to study our work from the categorical perspective of [20, 19, 6]. As a starting point one could try to rephrase our acceptance definition for streams (Definition 20) in the approach of [19].

References

- 1 Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, Elsevier, 1987. doi:10.1016/0890-5401(87)90052-6.
- 2 Dana Angluin, Udi Boker, and Dana Fisman. Families of dfas as acceptors of ω -regular languages. *Logical Methods in Computer Science*, Volume 14, Issue 1, February 2018. doi:10.23638/LMCS-14(1:15)2018.
- 3 Dana Angluin and Dana Fisman. Learning regular omega languages. In *Algorithmic Learning Theory*, volume 8776 of *Lecture Notes in Computer Science*, pages 125–139. Springer International Publishing, 2014. doi:10.1007/978-3-319-11662-4_10.
- 4 André Arnold. A syntactic congruence for rational ω -languages. *Theoretical Computer Science*, 39:333–335, Elsevier, 1985. doi:10.1016/0304-3975(85)90148-3.

- 5 Simone Barlocco, Clemens Kupke, and Jurriaan Rot. Coalgebra learning via duality. In *Foundations of Software Science and Computation Structures*, volume 11425 of *Lecture Notes in Computer Science*, pages 62–79. Springer International Publishing, 2019. doi:10.1007/978-3-030-17127-8_4.
- 6 Tomasz Brengos. A coalgebraic take on regular and omega-regular behaviour for systems with internal moves. In *International Conference on Concurrency Theory*, volume 118 of *LIPICs*, pages 25:1–25:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPICs.CONCUR.2018.25.
- 7 Hugues Calbrix, Maurice Nivat, and Andreas Podelski. Ultimately periodic words of rational ω -languages. In *Mathematical Foundations of Programming Semantics*, volume 802 of *Lecture Notes in Computer Science*, pages 554–566. Springer, 1993. doi:10.1007/3-540-58027-1_27.
- 8 Vincenzo Ciancia and Matteo Sammartino. A class of automata for the verification of infinite, resource-allocating behaviours. In *Trustworthy Global Computing*, volume 8902 of *Lecture Notes in Computer Science*, pages 97–111. Springer Berlin Heidelberg, 2014. doi:10.1007/978-3-662-45917-1_7.
- 9 Vincenzo Ciancia and Yde Venema. Stream automata are coalgebras. In *Coalgebraic Methods in Computer Science*, volume 7399 of *Lecture Notes in Computer Science*, pages 90–108. Springer, 2012. doi:10.1007/978-3-642-32784-1_6.
- 10 Szilárd Fazekas. Powers of regular languages. In Volker Diekert and Dirk Nowotka, editors, *Developments in Language Theory*, volume 5583 of *Lecture Notes in Computer Science*, pages 221–227. Springer, 2009. doi:10.1007/978-3-642-02737-6_17.
- 11 E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logic, and Infinite Games*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.
- 12 Yong Li, Yu-Fang Chen, Lijun Zhang, and Depeng Liu. A novel learning algorithm for büchi automata based on family of dfas and classification trees. In *Tools and Algorithms for the Construction and Analysis of Systems, Part I*, volume 10205 of *Lecture Notes in Computer Science*, pages 208–226, 2017. doi:10.1007/978-3-662-54577-5_12.
- 13 Christof Löding. Efficient minimization of deterministic weak ω -automata. *Information Processing Letters*, 79:105–109, Elsevier, 2001. doi:10.1016/S0020-0190(00)00183-6.
- 14 Oded Maler and Ludwig Staiger. On syntactic congruences for ω -languages. *Theoretical Computer Science*, 183:93–112, Elsevier, 1997. doi:10.1007/3-540-56503-5_58.
- 15 Dominique Perrin and Jean-Éric Pin. *Infinite Words: Automata, Semigroups, Logic and Games*. Elsevier, 2004.
- 16 J. Rutten. Automata and coinduction (an exercise in coalgebra). In *International Conference on Concurrency Theory*, *Lecture Notes in Computer Science*, pages 194–218. Springer, 1998. doi:10.1007/BFb0055624.
- 17 J. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249:3–80, Elsevier, 2000. doi:10.1016/S0304-3975(00)00056-6.
- 18 Ludwig Staiger. Finite-state ω -languages. *Journal of Computer and System Sciences*, 27:434–448, Elsevier, 1983. doi:10.1016/0022-0000(83)90051-X.
- 19 Natsuki Urabe and Ichiro Hasuo. Categorical büchi and parity conditions via alternating fixed points of functors. In *Coalgebraic Methods in Computer Science, Revised Selected Papers*, volume 11202 of *Lecture Notes in Computer Science*, pages 214–234. Springer, 2018. doi:10.1007/978-3-030-00389-0_12.
- 20 Natsuki Urabe, Shunsuke Shimizu, and Ichiro Hasuo. Coalgebraic trace semantics for büchi and parity automata. In *International Conference on Concurrency Theory*, volume 59 of *Leibniz International Proceedings in Informatics (LIPICs)*, pages 24:1–24:15, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPICs.CONCUR.2016.24.
- 21 Thomas Wilke. An algebraic theory for regular languages of finite and infinite words. *International Journal of Algebra and Computation*, 03(04):447–489, 1993. doi:10.1142/S0218196793000287.

A Proofs

Proof of Theorem 18. Let \mathbb{A} be lasso automaton. Decidability of coherence is straightforward, since it can be checked in terms of the equivalence of various pairs of initialized DFAs that can easily be constructed from the loop automaton of \mathbb{A} .

For circularity, it suffices to consider initialized DFAs. A routine argument shows that a language is circular iff $Pow(L) = L = Root(L)$, where the *power* and *root* of L are defined by $Pow(L) := \{u^k \mid u \in L, k \geq 1\}$ and $Root(L) := \{u \mid u^k \in L \text{ for some } k \geq 1\}$. The decidability of the equation, $L = Pow(L)$, was established by Fazekas [10].

The problem, whether $L(\mathbb{A}, a) = Root(L(\mathbb{A}, a_I))$ for a given initialized DFA (\mathbb{A}, a_I) , can be solved by first defining an initialized DFA (\mathbb{A}', i) that accepts $Root(L)$ and then checking language equivalence with L . So consider a DFA (A, δ, F) and a state $a_I \in A$. Let $\mathbb{A}' := (A^A, \theta, G)$ be the DFA where θ is given pointwise: $\theta(f, c) := \lambda a. \delta(fa, c)$. For the definition of G , define, for an arbitrary $g \in A^A$, the set $A_g := \{g^n a_I \mid n \geq 1\}$ – since A is finite, this set can be computed in at most $|A|$ steps. Put $G := \{g \in A^A \mid A_g \cap F \neq \emptyset\}$.

Let u an arbitrary finite word, and define $g_u := \widehat{\theta}(i, u)$. By construction we obtain for all $a \in A$ that $g_u a = \widehat{\delta}(a, u)$, so that $A_{g_u} = \{\widehat{\delta}(a_I, u^n) \mid n \geq 1\}$. Now consider the following chain of equivalences:

$$\begin{aligned}
 \mathbb{A}', i \triangleright u &\text{ iff } g_u = \widehat{\theta}(i, u) \in G && \text{(definition of acceptance)} \\
 &\text{ iff } A_{g_u} \cap F \neq \emptyset && \text{(definition } G) \\
 &\text{ iff } \widehat{\delta}(a_I, u^k) \in F, \text{ some } k \geq 1 && (A_{g_u} = \{\widehat{\delta}(a_I, u^n) \mid n \geq 1\}) \\
 &\text{ iff } u^k \in L = L(\mathbb{A}, a_I), \text{ some } k \geq 1 && \text{(definition of acceptance)} \\
 &\text{ iff } u \in Root(L) && \text{(definition } Root)
 \end{aligned}$$

From this it follows that $L(\mathbb{A}', i) = Root(L(\mathbb{A}, a_I))$, as required. \blacktriangleleft

Proof of Theorem 22. Let $\mathbb{P} = (P, \rho, \Pi)$ be a parity automaton, let $\mathbb{A}_{\mathbb{P}} = (P, X, \rho, \xi, \sigma, F)$ be its associated lasso automaton, and let N denote the range of Π . Fix a state $q \in P$.

For the inclusion $Streams(\mathbb{P}, q) \subseteq Streams(\mathbb{A}_{\mathbb{P}}, q)$, assume that $\mathbb{P}, q \triangleright \alpha$ for some stream α . Without loss of generality we may assume that α can be split as $\alpha = u\vec{v}$, such that, with $p := \widehat{\rho}(q, u)$, we have $p \xrightarrow{v_i} \rho p$ and $Inf(q, \alpha) = \rho^\circ(p, v_i)$, for each $i \in \omega$. Define $v_{ij} := v_i \cdots v_{j-1}$ for $i, j \in \omega$ with $i < j$. It follows by construction of $\mathbb{A}_{\mathbb{P}}$ that

$$\widehat{\sigma}:\widehat{\xi}(p, v_{ij}) \in F \text{ for each } i, j \text{ with } i < j. \quad (6)$$

Note that the equivalence relation on $\omega_{<} = \{(i, j) \in \omega^2 \mid i < j\}$ given by $(i, j) \sim (k, l)$ if $\widehat{\sigma}:\widehat{\xi}(p, v_{ij}) = \widehat{\sigma}:\widehat{\xi}(p, v_{kl})$, has finite index. It then follows by Ramsey's Theorem that there is an infinite subset $Y \subseteq \omega$, and a unique $z \in F$ such that $\widehat{\sigma}:\widehat{\xi}(p, v_{ij}) = z$, for each pair $i, j \in Y$ with $i < j$.

Enumerate $Y = \{k_0, k_1, \dots\}$ with $k_0 < k_1 < \dots$, and define $u' := v_{0k_0}$, for each $i \in \omega$, $w_i := v_{k_i k_{i+1}}$. It is obvious from these definitions that

$$\alpha = uu'\vec{w}. \quad (7)$$

In addition, we claim that

$$q \xrightarrow{uu'} \rho p \quad (8)$$

and that

$$p \xrightarrow{w_i} \rho p \text{ and } p \xrightarrow{w_i} \sigma:\xi z, \text{ for all } i \in \omega. \quad (9)$$

For the proof of (8) and (9), observe that $p \xrightarrow{v_j} p$ for each j , so that we find $p \xrightarrow{u'} p$ and $p \xrightarrow{w_i} p$ for each i on the basis of u' and each w_i being finite concatenations of v_j 's. From this we immediately obtain the first statement in (9), but also (8) because $q \xrightarrow{u} p \xrightarrow{u'} p$. The remaining, second, statement in (9) follows directly from the assumption on z and the definition of the w_j .

Finally, it follows directly from (7), (8) and (9) that α is accepted by (\mathbb{A}_P, q) .

For the inclusion $Streams(\mathbb{A}_{\mathbb{P}}, q) \subseteq Streams(\mathbb{P}, q)$, assume that $(\mathbb{A}_{\mathbb{P}}, q)$ accepts some given stream α . Then by definition we can split this stream as $\alpha = u\vec{v}$, and find states $p \in P$ and $t \in F$ such that $q \xrightarrow{u} p$ and $p \xrightarrow{v_i} p, p \xrightarrow{v_i} \sigma: \xi t$ for each $i \in \omega$. Fix some $i \in \omega$. By definition of $\mathbb{A}_{\mathbb{P}}$ it follows from $p \xrightarrow{v_i} \sigma: \xi t$ that $s_I^p \xrightarrow{v_i} \xi t$ in \mathbb{X}_p , where \mathbb{X}_p, s_I^p and ξ are as in Definition 12. But since it is straightforward to verify from $p \xrightarrow{v_i} p$ that $\rho^\circ(p, v_i)$ consists of all states traversed on the cycle $p \xrightarrow{v_i} p$, it follows that $t \in (P \times N)^P$ satisfies $t(p) = (p, \max(\Pi[\rho^\circ(p, v_i)]))$. Since t is accepting it follows that $\max(\Pi[\rho^\circ(p, v_i)])$ is even. Since this holds for each $i \in \omega$, it easily follows that (\mathbb{P}, q) accepts α . \blacktriangleleft

Proof of Theorem 23. Fix $\mathbb{A} = (P, X, \rho, \xi, \sigma, F)$ and $\mathbb{A}' = (P', X', \rho', \xi', \sigma', F')$, and let $h : \mathbb{A} \rightarrow \mathbb{A}'$ be an Ω -morphism. The proof of the inclusion $Streams(\mathbb{A}, q) \subseteq Streams(\mathbb{A}', q')$ is routine and left to the reader.

For the opposite inclusion, assume that (\mathbb{A}', hq) accepts some stream α . Let $u, (v_i)_{i \in \omega}, p' \in P'$ and $z' \in F'$ bear witness to this fact, in the sense that $\alpha = u\vec{v}$, $hq \xrightarrow{u} p'$ and $p' \xrightarrow{v_i} p', p' \xrightarrow{v_i} \sigma': \xi' z'$ for each $i \in \omega$.

Define $p_0 := \widehat{\rho}(q, u)$ and $p_{i+1} := \widehat{\rho}(p_i, v_i)$ for $i > 0$. Observe that $hp_i = p'$ for all $i \in \omega$. Since P is finite, so is the set $Q := \{p_i \mid i \in \omega\}$, and hence, some element p of Q is traversed infinitely often in the path $p_0 \xrightarrow{v_0} p_1 \xrightarrow{v_1} p_2 \cdots$. In other words, there is an infinite subset $K = \{k_0, k_1, \dots\} \subseteq \omega$ with $k_0 < k_1 < \dots$ such that $p_0 \xrightarrow{v_0 \cdots v_{k_0-1}} p$ and $p \xrightarrow{v_{k_i} \cdots v_{k_{i+1}-1}} p$ for all $i \in \omega$. Define $u' := v_0 \cdots v_{k_0-1}$ and $w_i := v_{k_i} \cdots v_{k_{i+1}-1}$, then we have $\alpha = uu'\vec{w}$, $q \xrightarrow{uu'} p$ and $p \xrightarrow{w_i} p$, for each $i \in \omega$.

Define, for $i < j \in \omega$, the word $w_{ij} := w_i \cdots w_{j-1}$ and the state $z_{ij} := \widehat{\sigma: \xi}(p, w_{ij})$. By Ramsey's Theorem there must be an infinite set $N \subseteq \omega$ and a single element $z \in X$ such that $z_{ij} = z$ for all $i, j \in N$. Note that $hz_{ij} = z'$ for all $i, j \in N$, since h is an Ω -morphism. Write $N = \{n_0, n_1, \dots\}$ with $n_0 < n_1 < \dots$, and define $u'' := w_0 \cdots w_{n_0-1}$ and $s_i := w_{n_i} \cdots w_{n_{i+1}-1}$, then clearly we have

$$\alpha = uu'u''\vec{s}. \quad (10)$$

Second, we claim that

$$q \xrightarrow{uu'u''} p. \quad (11)$$

To see this, note that u'' is a finite concatenation of w_i 's. Since $p \xrightarrow{w_i} p$ for each i , it follows that $p \xrightarrow{u''} p$, and hence we obtain (11) from $q \xrightarrow{uu'} p \xrightarrow{u''} p$. In addition, we have

$$p \xrightarrow{s_i} p \text{ and } p \xrightarrow{s_i} \sigma: \xi z, \text{ for all } i \in \omega. \quad (12)$$

Here the first statement follows from each s_i being a finite concatenation of w_j 's, and the second is by assumption on z .

Finally, the fact that $\mathbb{A}, q \triangleright \alpha$ is immediate from (10), (11) and (12). \blacktriangleleft

Proof of Corollary 24. It follows from Fact 17 that there is an initialized parity automaton (\mathbb{P}, p) such that $Lassos(\mathbb{A}, q) = Lassos(\mathbb{P}, p)$, and from Fact 14 that $Lassos(\mathbb{P}, p) =$

$Lassos(\mathbb{A}_{\mathbb{P}}, p)$. From this it is immediate that $Lassos(\mathbb{A}, q) = Lassos(\mathbb{A}_{\mathbb{P}}, p)$, and so Fact 10 yields that $\mathbb{A}, q \Leftrightarrow \mathbb{A}_{\mathbb{P}}, p$. Then we may derive by Proposition 8 and Theorem 23 that $Streams(\mathbb{A}, q) = Streams(\mathbb{A}_{\mathbb{P}}, p)$, and so by Theorem 22 we find that $Streams(\mathbb{A}, q) = Streams(\mathbb{P}, p)$. This immediately gives that $Streams(\mathbb{A}, q)$ is regular, and gathering our findings we obtain that $Lassos(Streams(\mathbb{A}, q)) = Lassos(Streams(\mathbb{P}, p)) = Lassos(\mathbb{P}, p) = Lassos(\mathbb{A}, q)$. ◀

Proof of Theorem 25. The direction from left to right is immediate by Proposition 8 and Theorem 23. The opposite direction follows from Corollary 24 and Fact 10. ◀

Proof of Theorem 28. The first part of the theorem is immediate by the fact that the quotient map from \mathbb{A} to $\mathbb{A}/\Leftrightarrow$ is an Ω -morphism. For part 2, assume that $Streams(\mathbb{A}', p') = L$. Then by Theorem 25 we have that $\mathbb{A}, p \Leftrightarrow \mathbb{A}', p'$, so that a routine argument shows that every state in $\mathbb{A}'_{p'}$ is bisimilar to some state in \mathbb{A} . This yields a natural map h from $\mathbb{A}'_{p'}$ to $\mathbb{A}/\Leftrightarrow$ such that $h(p') = \bar{p}$. We leave it for the reader to verify that this map is an Ω -morphism. ◀

Proof of Theorem 32. With $\mathbb{A} = (P, X, \rho, \xi, \sigma, F)$, define the maps $h_0 : P \rightarrow \mathcal{P}(C^* \times C^+)$ and $h_1 : X \rightarrow \mathcal{P}C^*$ by putting

$$\begin{aligned} h_0(p) &:= Lassos(\mathbb{A}, p), \\ h_1(x) &:= L((X, \xi, F), x). \end{aligned}$$

It is a routine exercise to verify that this is an Ω -morphism. For uniqueness, let $h' : \mathbb{A} \rightarrow \mathbb{Z}$ be an Ω -morphism. Then for every spoke state p of \mathbb{A} we find that $hp = Lassos(\mathbb{A}, p) = Lassos(\mathbb{Z}, h'p) = h'p$. ◀

Proof of Theorem 34. We confine ourselves to a sketch. The equivalence of (1) and (2) is a direct consequence of the Theorems 31 and 32, so it suffices to show that (2) \Leftrightarrow (3).

For this purpose, fix a lasso language L . We first show that, for any pair of words u_0, u_1 :

$$u_0 \equiv u_1 \text{ iff } \widehat{\zeta}_0(L, u_0) = \widehat{\zeta}_0(L, u_1). \quad (13)$$

Second, for all words u and pairs of nonempty words v_0, v_1 , with $L_u := \widehat{\zeta}_0(L, u) (= \{(u'v) \mid (uu', v) \in L\})$ we can prove:

$$v_0 \equiv_{[u]} v_1 \text{ iff } \widehat{\sigma} : \widehat{\zeta}_1(L_u, v_0) = \widehat{\sigma} : \widehat{\zeta}_1(L_u, v_1) \quad (14)$$

By the previous two steps we may conclude that $(u_0, v_0) \equiv (u_1, v_1)$ if and only if, starting from L in the spoke part of \mathbb{Z} , consuming either u_0 or u_1 takes us to the same state L' , and from L' , consuming either v_0 or v_1 , takes us to the same state L'' in the loop part of \mathbb{Z} . Now let Y_L be the set of spoke states reachable from L , and for $M \in Y_L$, let Y'_M be the set of loop states reachable from M . It then follows by the above observation that the equivalence classes of \equiv_L are in one-to-one correspondence with the set $\bigsqcup_{M \in Y_L} M$. This suffices to prove that \equiv_L has finite index iff L generates a finite subcoalgebra in \mathbb{Z} . ◀

Proof of Proposition 36. Fix a stream $\alpha = \vec{v}$. By lasso determinacy of L there is an infinite set $Y \subseteq \omega$ as in the definition. Write $Y = \{n_0, n_1, \dots\}$ with $n_0 < n_1 < \dots$. Define $w_0 := v_0 \cdots v_{n_0}$ and $w_{i+1} := v_{n_i+1} \cdots v_{n_{i+1}}$. Then $\alpha = \vec{w}$ and for all $j, k \in \omega$ with $j < k$ we have

$$\alpha \in L \iff (w_0 \cdots w_j)(w_{j+1} \cdots w_k)^\omega \in L \quad (15)$$

Now by the LD property of L' there is an infinite set $Y' \in \omega$ as in the definition. Take any two elements $j, k \in Y'$ with $j < k$. Then we have

$$\alpha \in L' \iff (w_0 \cdots w_j)(w_{j+1} \cdots w_k)^\omega \in L' \quad (16)$$

It is then immediate by (15), (16) and the assumption $L_{\text{ssos}}(L) = L_{\text{ssos}}(L')$ that


$$\alpha \in L \iff \alpha \in L'.$$

Since α was arbitrary, this shows that $L = L'$. \blacktriangleleft

Proof of Theorem 37. The equivalence of (2) and (3) is immediate by Theorem 34, and the implication from (1) to (2/3) follows from the same result, together with the observation that regular ω -languages are lasso-determined.

For the remaining implication (2/3 \Rightarrow 1), assume (2), and let L' be the stream language accepted by the pointed Ω -coalgebra $(\mathbb{Z}, L_{\text{ssos}}(L))$. It follows that $L_{\text{ssos}}(L) = L_{\text{ssos}}(L')$, and since L' , being regular, is lasso-determined, Proposition 36 implies $L = L'$, which immediately yields the regularity of L . \blacktriangleleft

Tree Automata as Algebras: Minimisation and Determinisation

Gerco van Heerdt 

University College London, United Kingdom
gerco.heerdt@ucl.ac.uk

Tobias Kappé 


University College London, United Kingdom
tkappe@cs.ucl.ac.uk

Jurriaan Rot

University College London, United Kingdom
Radboud University, Nijmegen, The Netherlands
jrot@cs.ru.nl

Matteo Sammartino 

University College London, United Kingdom
m.sammartino@ucl.ac.uk

Alexandra Silva 

University College London, United Kingdom
alexandra.silva@ucl.ac.uk

Abstract

We study a categorical generalisation of tree automata, as algebras for a fixed endofunctor endowed with initial and final states. Under mild assumptions about the base category, we present a general minimisation algorithm for these automata. We then build upon and extend an existing generalisation of the Nerode equivalence to a categorical setting and relate it to the existence of minimal automata. Finally, we show that generalised types of side-effects, such as non-determinism, can be captured by this categorical framework, leading to a general determinisation procedure.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases tree automata, algebras, minimisation, determinisation, Nerode equivalence

Digital Object Identifier 10.4230/LIPIcs.CALCO.2019.6

Funding This work was partially supported by the ERC Starting Grant ProFoundNet (grant code 679127), a Leverhulme Prize (PLP-2016-129) and a Marie Curie Fellowship (grant code 795119).

1 Introduction

Automata have been extensively studied using category theory, both from an algebraic and a coalgebraic perspective [26, 6, 43, 40]. Categorical insights have enabled the development of generic algorithms for minimisation [4], determinisation [45], and equivalence checking [15].

A fruitful line of work has focussed on characterising the semantics of different types of automata as final coalgebras. The final coalgebra contains unique representatives of behaviour, and the existence of a minimal automaton can be formalised by a suitable factorisation of the map from a given automaton into the final coalgebra. Algorithms to compute the minimal automaton can be devised based on the final sequence, which yields procedures resembling classical partition refinement [32, 18]. Unfortunately, bottom-up tree automata do not fit the abstract framework of final coalgebras.¹ This impeded the application of abstract

¹ The language semantics of top-down tree automata represented as coalgebras is given in [30], based on a transformation to bottom-up tree automata. In this paper, we focus on bottom-up automata only.



algorithms for minimisation, determinisation, and equivalence. We embrace the categorical *algebraic* view on automata due to Arbib and Manes [8] to study bottom-up tree automata (Section 3). This algebraic approach is also treated in detail by Adámek and Trnková [6], who, among other results, give conditions under which minimal realisations exist (see also [3]) and constructions to determinise partial and non-deterministic bottom-up tree automata. However, generic *algorithms* for minimisation, and a more abstract and uniform picture of determinisation, have not been studied in this context.

The contributions of this paper are three-fold. First, we explore the notion of *cobase* to devise an iterative construction for minimising tree automata, at the abstract level of algebras, resembling partition refinement (Section 4). The notion of cobase is dual to that of base [11], which plays a key role in reachability of coalgebras [46, 10] and therefore in minimisation of automata. Second, we study a different characterisation of minimality via the Nerode equivalence, again based on work of Arbib and Manes [8], and provide a generalisation using monads that allows to treat automata with equations (Section 5). Third, we extend bottom-up tree automata to algebras in the Kleisli category of a monad, which enables us to study tree automata enriched with side-effects and derive an associated determinisation procedure (Section 6). We demonstrate the generality of our approach by applying it both to classical examples – deterministic, non-deterministic, and multiplicity/weighted tree automata – and to a novel kind of tree automata, namely *nominal* tree automata.

2 Preliminaries

We assume basic knowledge of category theory. Throughout this paper, we fix a category \mathbf{C} .

Monads. A *monad* on \mathbf{C} is a triple (T, η, μ) consisting of an endofunctor T on \mathbf{C} and two natural transformations: a *unit* $\eta: \text{Id} \Rightarrow T$ and a *multiplication* $\mu: T^2 \Rightarrow T$, which satisfy the compatibility laws $\mu \circ \eta_T = \text{id}_T = \mu \circ T\eta$ and $\mu \circ \mu_T = \mu \circ T\mu$.

► **Example 2.1.** The triple $(\mathcal{P}_f, \{-\}, \cup)$ is a monad on \mathbf{Set} , where \mathcal{P}_f is the finite powerset functor, $\{-\}$ is the singleton operation, and \cup is union of sets. Another example is the *multiplicity monad* $(\mathcal{M}_{\mathbb{F}}, e, m)$ for a field \mathbb{F} , where $\mathcal{M}_{\mathbb{F}}$ is the functor sending a set X to $\mathcal{M}_{\mathbb{F}}X = \{\varphi: X \rightarrow \mathbb{F} \mid \varphi \text{ has finite support}\}$. An element φ can be seen as a formal finite sum $\sum_i s_i x_i$, where each x_i has multiplicity s_i . The unit e sends x to $1x$ and the multiplication is $m_X(\sum_i s_i \varphi_i)(x) = \sum_i s_i \cdot \varphi_i(x)$, where \cdot is the field multiplication.

Algebras and Varietors. We fix a functor $\Sigma: \mathbf{C} \rightarrow \mathbf{C}$ and write $\mathbf{Alg}(\Sigma)$ for the category of Σ -algebras. Throughout this paper, we assume that Σ is a *variator* [6], i.e., that the forgetful functor $U: \mathbf{Alg}(\Sigma) \rightarrow \mathbf{C}$ admits a left adjoint $F: \mathbf{C} \rightarrow \mathbf{Alg}(\Sigma)$. The variator Σ induces a monad $(\Sigma^\circ, \eta, \mu)$ on \mathbf{C} , where $\Sigma^\circ = UF$. Given an object X of \mathbf{C} , we refer to $FX = (\Sigma^\circ X, \alpha_X)$ as the *free Σ -algebra* over X . The free Σ -algebra satisfies the following: for every Σ -algebra Q and every morphism $x: X \rightarrow UQ$ of \mathbf{C} , there is a unique Σ -algebra morphism $x^\#: (\Sigma^\circ X, \alpha_X) \rightarrow Q$ with $U(x^\#) \circ \eta_X = x$.

► **Example 2.2.** A functor is *finitary* if it preserves filtered colimits. Any finitary \mathbf{Set} functor is a variator: free algebras over a set X can be obtained as a colimit of a transfinite sequence [2, 28]. A *polynomial functor* on \mathbf{Set} is a functor inductively defined by $P := \text{id} \mid A \mid P_1 \times P_2 \mid \coprod_{i \in I} P_i$ where A is any constant functor. Polynomial functors are finitary and therefore variators.

3 Tree automata, categorically

In this section we start our categorical investigation of (bottom-up) tree automata. We first discuss a general notion of automaton over an endofunctor Σ due to Arbib and Manes [8] and then discuss how this notion can be instantiated to obtain various kinds of automata.

► **Definition 3.1** (Σ -tree automaton). *A Σ -tree automaton over objects I and O in \mathcal{C} is a tuple (Q, δ, i, o) such that (Q, δ) is a Σ -algebra and $i: I \rightarrow Q$ and $o: Q \rightarrow O$ are morphisms of \mathcal{C} . The objects I and O are referred to as the input object and output object respectively. A homomorphism from an automaton (Q, δ, i, o) to an automaton (Q', δ', i', o') is a Σ -algebra homomorphism $h: Q \rightarrow Q'$ (i.e., $\delta' \circ \Sigma h = h \circ \delta$) such that $h \circ i = i'$ and $o' \circ h = o$.*

Throughout this paper, we fix input and output objects I and O respectively. If Σ is clear from the context we sometimes refer to a Σ -tree automaton simply as an automaton.

A *language* (over Σ) is a morphism $L: \Sigma^\circ I \rightarrow O$. In the context of an automaton $\mathcal{A} = (Q, \delta, i, o)$, we can think of $U(i^\sharp): \Sigma^\circ I \rightarrow Q$, induced by the free Σ -algebra FI on I , as the *reachability map*, telling us which state is reached by parsing an element of the free algebra over I . We will write $\text{rch}_{\mathcal{A}}$ (or rch if \mathcal{A} is obvious) for $U(i^\sharp)$. The *language of \mathcal{A}* is the morphism $\mathcal{L}(\mathcal{A}): \Sigma^\circ I \rightarrow O$ in \mathcal{C} given by $\mathcal{L}(\mathcal{A}) = o \circ \text{rch}_{\mathcal{A}}$.

► **Example 3.2** (Deterministic bottom-up tree automata). Let us see how Σ -tree automata can capture deterministic bottom-up tree automata. We first recall some basic concepts.

A *ranked alphabet* is a finite set of symbols Γ , where each $\gamma \in \Gamma$ is equipped with an *arity* $\text{ar}(\gamma) \in \mathbb{N}$. A *frontier alphabet* is a finite set of symbols I . The set of Γ -trees over I , denoted $\mathcal{T}_\Gamma(I)$, is the smallest set such that $I \subseteq \mathcal{T}_\Gamma(I)$, and for all $\gamma \in \Gamma$ we have that $t_1, \dots, t_{\text{ar}(\gamma)} \in \mathcal{T}_\Gamma(I)$ implies $(\gamma, t_1, \dots, t_{\text{ar}(\gamma)}) \in \mathcal{T}_\Gamma(I)$. In other words, $\mathcal{T}_\Gamma(I)$ consists of finite trees with leaves labelled by symbols from I and internal nodes labelled by symbols from Γ ; the number of children of each internal node matches the arity of its label.

A ranked alphabet Γ gives rise to a polynomial *signature endofunctor* $\Sigma: \text{Set} \rightarrow \text{Set}$ given by $\Sigma X = \coprod_{\gamma \in \Gamma} X^{\text{ar}(\gamma)}$. A *deterministic bottom-up tree automaton* is a Σ -tree automaton $\mathcal{A} = (Q, \delta, i, o)$ where Q is finite, Σ is a signature endofunctor, and $O = 2$. Here Q is the set of *states*, $i: I \rightarrow Q$ is the *initial assignment*, $o: Q \rightarrow 2$ is the characteristic function of *final states*, and for each $\gamma \in \Gamma$ we have a transition function $\delta_\gamma = \delta \circ \kappa_\gamma: Q^{\text{ar}(\gamma)} \rightarrow Q$. The *language $\mathcal{L}(\mathcal{A})$* is the set of all Γ -trees t such that $(o \circ \hat{\delta})(t) = 1$, where $\hat{\delta}: \mathcal{T}_\Gamma(I) \rightarrow Q$ extends δ to trees by structural recursion:

$$\hat{\delta}(\ell) = i(\ell) \quad (\ell \in I) \qquad \hat{\delta}(\gamma, t_1, \dots, t_k) = \delta_\gamma(\hat{\delta}(t_1), \dots, \hat{\delta}(t_k))$$

In other words, $\mathcal{L}(\mathcal{A})$ contains the trees that evaluate to a final state. The map $\hat{\delta}$ above is the transpose i^\sharp in the relevant adjunction between Set and $\text{Alg}(\Sigma)$, where the left adjoint sends a set I to the Σ -algebra with carrier $\mathcal{T}_\Gamma(I)$ and the obvious structure map.

3.1 Nominal tree automata

To show the versatility of our definition, we instantiate it in the category Nom of nominal sets and equivariant functions. This results in a notion of nominal tree automaton – along the lines of nominal automata theory [13] – which, as we shall see below, provides a useful model for languages of trees with variables and variable binding. We first recall some basic notations of nominal set theory [41]. Let \mathbb{A} be a countable set of *atoms*, and let $\text{Sym}(\mathbb{A})$ be the associated symmetry group, consisting of all permutations on \mathbb{A} . A *nominal set* is a pair (X, \cdot) of a set X and a function $\cdot: \text{Sym}(\mathbb{A}) \times X \rightarrow X$ forming a left action of $\text{Sym}(\mathbb{A})$ on X .

Each $x \in X$ is required to have *finite support*, i.e., there must exist a finite $A \subseteq \mathbb{A}$ such that for all $\pi \in \text{Sym}(\mathbb{A})$, if π is equal to id_A when restricted to A , then $\pi \cdot x = x$. The minimal such A is denoted $\text{supp}(x)$, and can be understood as the set of “free” names of x . Given $x \in X$, its *orbit* is the set $\{\pi \cdot x \mid \pi \in \text{Sym}(\mathbb{A})\}$. We say that a nominal set X is *orbit-finite* whenever it has finitely many orbits. An *equivariant function* $f: (X, \cdot) \rightarrow (Y, \cdot)$ is a function $X \rightarrow Y$ that respects permutations, i.e., $f(\pi \cdot x) = \pi \cdot f(x)$.

Polynomial functors in Nom support additional operations [20], such as the *name abstraction* functor $[\mathbb{A}]: \text{Nom} \rightarrow \text{Nom}$, which “binds” a name in the support. For instance, if $x \in X$, then $\langle a \rangle x \in [\mathbb{A}]X$, with $\text{supp}(\langle a \rangle x) = \text{supp}(x) \setminus \{a\}$. The element $\langle a \rangle x$ should be thought of as an equivalence class *up to α -conversion* w.r.t. the binder $\langle a \rangle$. We can then define tree automata for parsing trees with binders. Consider for instance $\Sigma_\lambda: \text{Nom} \rightarrow \text{Nom}$ given by

$$\Sigma_\lambda X = \underbrace{X \times X}_{\text{appl}} + \underbrace{[\mathbb{A}]X}_{\text{lambda}}$$

describing the syntax of the λ -calculus [21]. This functor is finitary [20], which implies the existence of free algebras. Fixing $I = \mathbb{A}$, the carrier of the free Σ_λ -algebra over I consists of parse trees for λ -terms (up to α -conversion) with variables in \mathbb{A} . We can then define automata parsing these trees as $\mathcal{A} = (Q, \delta, i, o)$, where

- Q is a nominal set;
- δ consists of two equivariant functions $\delta_{\text{appl}}: Q \times Q \rightarrow Q$ and $\delta_{\text{lambda}}: [\mathbb{A}]Q \rightarrow Q$;
- $i: \mathbb{A} \rightarrow Q$ is an equivariant function, selecting states for parsing variables;
- $o: Q \rightarrow 2$ is an equivariant characteristic function of final states, which implies that if a state is final, so are all the states in its orbit.

The reachability function is the equivariant function given by

$$\text{rch}_{\mathcal{A}}(t) = \begin{cases} i(t) & \text{if } t \in \mathbb{A} \\ \delta_{\text{appl}}(\text{rch}_{\mathcal{A}}(t_1), \text{rch}_{\mathcal{A}}(t_2)) & \text{if } t = (t_1, t_2) \\ \delta_{\text{lambda}}(\langle a \rangle \text{rch}_{\mathcal{A}}(t')) & \text{if } t = \langle a \rangle t'. \end{cases}$$

The most interesting case is the last one: in order to parse the α -equivalence class $\langle a \rangle t'$, we first parse any tree t' such that $\langle a \rangle t'$ is in the class, and then we take the resulting state up to α -conversion w.r.t. $\langle a \rangle$. Note that $\mathcal{L}(\mathcal{A})$ is equivariant, i.e., invariant under permutations of atoms. Thus \mathcal{A} recognises λ -trees up to bijective renamings of variables.

4 Minimisation

In this section we define a construction that allows to minimise a given tree automaton. We start with a few basic preliminary notions related to quotients and factorisation systems.

Factorisations. An $(\mathcal{E}, \mathcal{M})$ -*factorisation system* on \mathbf{C} consists of classes of morphisms \mathcal{E} and \mathcal{M} , closed under composition with isos, such that for every morphism f in \mathbf{C} there exist $e \in \mathcal{E}$ and $m \in \mathcal{M}$ with $f = m \circ e$, and we have a unique diagonal fill-in property.

We list a few properties of factorisation systems. First, both \mathcal{E} and \mathcal{M} are closed under composition. Furthermore, if $g \circ f \in \mathcal{E}$ and $f \in \mathcal{E}$, then $g \in \mathcal{E}$. Lastly, if \mathcal{E} consists of epimorphisms, then it is closed under cointersections, i.e., wide pushouts of epimorphisms [5]. A functor Σ is said to *preserve \mathcal{E} -cointersections* if it preserves wide pushouts of epimorphisms in \mathcal{E} . In that case, for an epimorphism e , if $e \in \mathcal{E}$ then Σe is again an epimorphism.

Quotients. Define by \leq the order on morphisms with common domain given by $f \leq g$ iff $\exists h.g = h \circ f$. This induces an equivalence relation on such morphisms. A *quotient* of an object X is an epimorphism $q: X \twoheadrightarrow X'$ identified up to the equivalence, i.e., an equivalence class. We denote by $\text{Quot}(X)$ the class of all quotients of X . The underlying category \mathbf{C} is said to be *cowellpowered* if $\text{Quot}(X)$ is a set for every X . In that case, if \mathbf{C} is cocomplete, $\text{Quot}(X)$ forms a complete lattice, with the order given by \leq , and the least upper bound (join) given by cointersection. We denote by $\text{Quot}_{\mathcal{E}}(X)$ the set of quotients of X that are in \mathcal{E} . (This is well-defined because \mathcal{E} is closed under isomorphisms.)

► **Assumption 4.1.** Throughout this section, \mathbf{C} is cocomplete and cowellpowered. Moreover, we fix an $(\mathcal{E}, \mathcal{M})$ -factorisation system in \mathbf{C} , where \mathcal{E} contains epimorphisms only.

► **Remark 4.2.** The category Set is cocomplete and cowellpowered, and so is Nom introduced in Section 3.1. In general, the existence of an (epi, strong mono)-factorisation system already follows from \mathbf{C} being cocomplete and cowellpowered [16]. Allowing a more general choice of factorisation system will be useful in Section 5, where we work with a different \mathcal{E} .

Let (Q, δ) be a Σ -algebra. A *quotient algebra* is a Σ -algebra (Q', δ') together with a quotient $q: Q \twoheadrightarrow Q'$ in \mathcal{E} that is an algebra homomorphism. Given a Σ -tree automaton (Q, δ, i, o) , a *quotient automaton* is a Σ -tree automaton (Q', δ', i', o') together with a quotient $q: Q \twoheadrightarrow Q'$ in \mathcal{E} that is a homomorphism of automata.

► **Definition 4.3 (Minimisation).** *The minimisation of a Σ -tree automaton (Q, δ, i, o) is a quotient automaton $(Q_m, \delta_m, i_m, o_m)$, $q: Q \twoheadrightarrow Q_m$, such that for any quotient automaton (Q', δ', i', o') , $q': Q \twoheadrightarrow Q'$ of (Q, δ, i, o) there exists a (necessarily unique) automaton homomorphism $h: Q' \twoheadrightarrow Q_m$ such that $h \circ q' = q$.*

Minimisation is called *minimal reduction* in [6]. Note that the morphism h in the definition of minimisation is in \mathcal{E} , since q' and q are. In the sequel, we sometimes refer to a quotient $q: Q \twoheadrightarrow Q_m$ as the minimisation if there exist δ_m, i_m, o_m turning $(Q_m, \delta_m, i_m, o_m), q$ into the minimisation of (Q, δ, i, o) .

► **Definition 4.4.** *A Σ -tree automaton \mathcal{A} is said to be reachable if the associated reachability map rch is in \mathcal{E} . It is minimal if it is reachable and for every reachable Σ -tree automaton \mathcal{A}' s.t. $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$ there exists a (necessarily unique) homomorphism from \mathcal{A}' to \mathcal{A} .*

The above definition of minimality relies on reachability; a more orthogonal (but equivalent) definition of minimality is explored in Section 4.2.

We conclude with a few observations on the connection between minimisation and minimality, treated in detail in [6]. We say Σ *admits minimisation* of reachable automata if every reachable automaton over Σ has a minimisation.

► **Lemma 4.5.** *An automaton \mathcal{A} is minimal iff it is the minimisation of $(\Sigma^\circ I, \alpha_I, \eta_I, \mathcal{L}(\mathcal{A}))$.*

► **Lemma 4.6.** *The functor Σ admits minimisation of reachable automata if and only if a minimal automaton exists for every language over Σ . In that case, if \mathcal{A} is reachable, then the minimisation of \mathcal{A} is minimal.*

Proof. For the equivalence, the implication left to right follows from Lemma 4.5. For the converse, one readily shows that the minimisation of an automaton \mathcal{A} is given by the minimal automaton accepting $\mathcal{L}(\mathcal{A})$. The second statement holds by uniqueness of minimisations. ◀

4.1 Minimisation via the cobase

We show how to compute the minimisation of a given automaton (Q, δ, i, o) using the so-called *cobase* [11]. This is the dual of the base, which is used in [10, 46] for reachability of coalgebras. The cobase allows us to characterise the minimisation as the greatest fixed point of a certain monotone operator on $\text{Quot}_{\mathcal{E}}(Q)$, which is a complete lattice by Assumption 4.1.

► **Definition 4.7.** *Let $f: \Sigma X \rightarrow Y$ be a morphism. The (\mathcal{E}) -cobase of f (if it exists) is the greatest quotient $q \in \text{Quot}_{\mathcal{E}}(X)$ such that there exists a morphism g with $g \circ \Sigma q = f$.*

A concrete instance of the cobase will be given below in Example 4.11. The cobase can be computed as the join of all quotients satisfying the relevant condition, provided that the functor preserves cointersections.

► **Theorem 4.8** (Existence of cobases). *Suppose $\Sigma: \mathbf{C} \rightarrow \mathbf{C}$ preserves \mathcal{E} -cointersections. Then every map $f: \Sigma X \rightarrow Y$ has an \mathcal{E} -cobase, given by the cointersection*

$$\bigvee \{q \in \text{Quot}_{\mathcal{E}}(X) \mid \exists g. g \circ \Sigma q = f\}.$$

Proof. For \mathcal{E} the class of all epis, the dual is shown in [10, 46]. The proof goes through in the current, more general setting, using that \mathcal{E} is closed under cointersections. ◀

► **Remark 4.9.** A Set functor preserves cointersections iff it is finitary [6]. In particular, this is the case for polynomial functors. For **Nom** functors we can use that, in general, a functor preserves cointersections if it is finitary and preserves reflexive coequalisers [6]. These conditions hold for polynomial **Nom** functors introduced in Section 3.1, because they preserve sifted colimits [34], which include filtered colimits and reflexive coequalisers.

We now define an operator on quotients of the state space of an automaton, which characterises the minimisation of an automaton and gives a way of computing it. To this end, given a Σ -algebra (Q, δ) and a quotient $q: Q \twoheadrightarrow Q' \in \text{Quot}_{\mathcal{E}}(Q)$, define the quotient $\Theta_{\delta}(q): Q \twoheadrightarrow \Theta_{\delta}(Q')$ as the cobase of $q \circ \delta$. This defines a monotone operator $\Theta_{\delta}: \text{Quot}_{\mathcal{E}}(Q) \rightarrow \text{Quot}_{\mathcal{E}}(Q)$ that has the following important property (see [10, 46]):

► **Lemma 4.10.** *Suppose Σ preserves \mathcal{E} -cointersections. For any Σ -algebra (Q, δ) , a quotient $q: Q \twoheadrightarrow Q'$ in $\text{Quot}_{\mathcal{E}}(Q)$ satisfies $q \leq \Theta_{\delta}(q)$ iff there is an algebra structure $\delta': \Sigma Q' \rightarrow Q'$ turning q into an algebra homomorphism.*

The operator Θ_{δ} allows us to quotient the transition structure of the automaton. In order to obtain the minimal automaton, we incorporate the output map $o: Q \rightarrow O$ into the construction of a monotone operator based on Θ_{δ} . For technical convenience, we assume that this map is an element of $\text{Quot}_{\mathcal{E}}(Q)$.² The relevant monotone operator for minimisation is $\Theta_{\delta} \wedge o$ (where the meet \wedge is taken pointwise in $\text{Quot}_{\mathcal{E}}(Q)$).

► **Example 4.11.** Let $\Sigma: \mathbf{Set} \rightarrow \mathbf{Set}$ be a polynomial functor induced by signature Γ . We first spell out what the cobase means concretely in this case and then study the operator Θ_{δ} in more detail. Since Σ is an endofunctor on **Set**, the cobase of a map $f: \Sigma X \rightarrow Y$ is the largest quotient $q \in \text{Quot}_{\mathcal{E}}(X)$ such that for all $t, t' \in \Sigma X$:

$$\text{if } \Sigma q(t) = \Sigma q(t'), \text{ then } f(t) = f(t').$$

² This is not a real restriction: one can just pre-process the automaton by factorising o , i.e., keeping only those outputs actually occurring in the automaton.

This means that for every $\gamma \in \Gamma$ with $k = \text{ar}(\gamma)$, and any $x_1, \dots, x_k, y_1, \dots, y_k$, we have that

$$\frac{q(x_1) = q(y_1) \quad \dots \quad q(x_k) = q(y_k)}{f(\kappa_\gamma(x_1, \dots, x_k)) = f(\kappa_\gamma(y_1, \dots, y_k))}$$

or equivalently that for all x_1, \dots, x_k and x'_i with $1 \leq i \leq k$ we have

$$\frac{q(x_i) = q(x'_i)}{f(\kappa_\gamma(x_1, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_k)) = f(\kappa_\gamma(x_1, \dots, x_{i-1}, x'_i, x_{i+1}, \dots, x_k))}$$

Suppose (Q, δ, i, o) is an automaton. For $q \in \text{Quot}(Q)$, we have $q \leq \Theta_\delta(q) \wedge o$ iff

- for all $x, x' \in Q$: if $q(x) = q(x')$, then $o(x) = o(x')$; and
- for all $\gamma \in \Gamma$ with $k = \text{ar}(\gamma)$, and x_1, \dots, x_k and x'_i with $1 \leq i \leq k$ we have

$$\frac{q(x_i) = q(x'_i)}{q(\delta_\gamma(x_1, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_k)) = q(\delta_\gamma(x_1, \dots, x_{i-1}, x'_i, x_{i+1}, \dots, x_k))}$$

A partition q with the above two properties is known as a *forward bisimulation* [25].

► **Theorem 4.12.** *Suppose Σ preserves \mathcal{E} -cointersections. Let (Q, δ, i, o) be an automaton, where $o \in \text{Quot}_\mathcal{E}(Q)$. Then $\text{gfp}(\Theta_\delta \wedge o)$ is the minimisation of (Q, δ, i, o) .*

Proof. Denote the quotient $\text{gfp}(\Theta_\delta \wedge o)$ by $q_m: Q \twoheadrightarrow Q_m$. Thus $q_m \leq \Theta_\delta(q_m)$ and $q_m \leq o$, hence (using Lemma 4.10) there exist δ_m, o_m turning q_m into an automaton homomorphism from (Q, δ, i, o) to $(Q_m, \delta_m, q_m \circ i, o_m)$. We show that this is the minimisation of (Q, δ, i, o) .

To this end, let (Q', δ', i', o') , $q': Q \twoheadrightarrow Q'$ be a quotient automaton of (Q, δ, i, o) . By Lemma 4.10 we get $q' \leq \Theta_\delta(q')$, and since $o' \circ q' = o$ we have $q' \leq o$, hence $q' \leq \Theta_\delta(q') \wedge o$. Thus $q' \leq \text{gfp}(\Theta_\delta \wedge o)$, i.e., there is a quotient $h: Q' \twoheadrightarrow Q_m$ such that $h \circ q' = q_m$. It only remains to show that h is a homomorphism of automata. First, since $q' \in \mathcal{E}$ and Σ preserves \mathcal{E} -cointersections, $\Sigma q'$ is an epimorphism. Combined with the fact that q' and q_m are algebra homomorphisms and that $h \circ q' = q_m$, it easily follows that h is an algebra homomorphism. To see that it preserves the output, we have $o_m \circ h \circ q' = o_m \circ q_m = o = o' \circ q'$; hence, since q' is epic, we get $o_m \circ h = o'$. For preservation of the input, we have $h \circ i' = h \circ q' \circ i = q_m \circ i$, where the first step holds because q' is a homomorphism of automata. ◀

The above characterisation of minimisation of an automaton (Q, δ, i, o) gives us two ways of constructing it by standard lattice-theoretic computations. First, via the Knaster-Tarski theorem, we obtain it as the join of all post-fixed points of $\Theta_\delta \wedge o$, which, by Lemma 4.10, amounts to the join of all quotient algebras respecting the output map o . That corresponds to the construction in [6]. Second, and perhaps most interestingly, we obtain the minimisation of (Q, δ, i, o) by iterating $\Theta_\delta \wedge o$, starting from the top element \top of the lattice $\text{Quot}_\mathcal{E}(Q)$. The latter construction is analogous to the classical partition refinement algorithm: Starting from \top corresponds to identifying all states as equivalent (or in other words, starting from the coarsest equivalence class of states). Every iteration step of $\Theta_\delta \wedge o$ splits the states that can be distinguished successively by just outputs, trees of depth 1, trees of depth 2, etc. If the state space is finite, this construction terminates, yielding the minimisation of the original automaton by Theorem 4.12.

4.2 Simple automata

We defined an automaton to be minimal if it is reachable and satisfies a universal property w.r.t. reachable automata accepting the same language. It is also interesting to ask whether

there is another property that, together with reachability, implies minimality, but is not itself dependent on reachability [9]. Here we propose precisely such a condition.

► **Definition 4.13.** *An automaton (Q, δ, i, o) is called simple if for every quotient automaton (Q', δ', i', o') the associated quotient $q: Q \rightarrow Q'$ is an isomorphism.*

The result below asserts that minimal automata are precisely the automata that are simple and reachable. It can be seen as a refinement (and dual) of [10, Theorem 17], computing the reachable part of a coalgebra. One of the implications makes use of Theorem 4.12, so we assume that Σ preserves \mathcal{E} -cointersections.

► **Proposition 4.14.** *Suppose Σ preserves \mathcal{E} -cointersections. Let (Q, δ, i, o) be an automaton with $o \in \mathcal{E}$, and let (Q', δ', i', o') , $q: Q \rightarrow Q'$ be a quotient automaton. Then (Q', δ', i', o') is simple if and only if it is the minimisation of (Q, δ, i, o) .*

Proof. By Theorem 4.12, the minimisation of (Q, δ, i, o) exists. We denote it by $q_m: Q \rightarrow Q_m$ and its associated automaton structure by $(Q_m, \delta_m, i_m, o_m)$.

Suppose (Q', δ', i', o') is simple. Since Q_m is the minimisation of Q and Q' is a quotient automaton of Q , there exists a homomorphism of automata $h: Q' \rightarrow Q_m$. Since Q' is simple, this homomorphism is an iso.

Conversely, suppose (Q', δ', i', o') is the minimisation of (Q, δ, i, o) and consider any quotient automaton Q'' of Q' , witnessed by some $q': Q' \rightarrow Q''$. Then Q'' is also a quotient automaton of Q , via $q' \circ q$. Because Q' is the minimisation of Q , there exists $k: Q'' \rightarrow Q'$ such that $k \circ q' \circ q = q$. Thus $k \circ q' = \text{id}$, using that q is an epi. Since $q' \circ k \circ q' = q'$ and q' is an epi as well, we also have $q' \circ k = \text{id}$. Hence q' is an iso, as needed. ◀

► **Corollary 4.15.** *If Σ preserves \mathcal{E} -cointersections, then an automaton $\mathcal{A} = (Q, \delta, i, o)$ with $o \in \mathcal{E}$ is minimal if and only if it is simple and reachable.*

Proof. First, suppose that \mathcal{A} is minimal. By Lemma 4.5, we know that \mathcal{A} is the minimisation (and, in particular, a quotient) of $(\Sigma^\circ I, \alpha_I, \eta_I, \mathcal{L}(\mathcal{A}))$. In that case, \mathcal{A} is reachable, and thus, since $o \in \mathcal{E}$, we have $\mathcal{L}(\mathcal{A}) \in \mathcal{E}$. By Proposition 4.14, we conclude that \mathcal{A} is simple.

Conversely, let \mathcal{A} be simple and reachable. By reachability, \mathcal{A} is a quotient automaton of $(\Sigma^\circ I, \alpha_I, \eta_I, \mathcal{L}(\mathcal{A}))$; also, $\mathcal{L}(\mathcal{A}) = o \circ \text{rch} \in \mathcal{E}$. Proposition 4.14 then tells us that \mathcal{A} is the minimisation of $(\Sigma^\circ I, \alpha_I, \eta_I, \mathcal{L}(\mathcal{A}))$; by Lemma 4.5 we conclude that \mathcal{A} is minimal. ◀

5 Nerode equivalence

We now show a generalised Nerode equivalence from which the minimal automaton can be constructed. Most of this section is based upon the work by Arbib and Manes [8], whose construction was further studied and refined by Anderson et al. [7] and Adámek and Trnková [6]. We make a significant improvement in generality by phrasing the central equivalence definition (Definition 5.5) in terms of an arbitrary monad, which unlike the previous cited work allows applications to algebras satisfying a fixed set of equations. A monad generalisation of the Myhill-Nerode theorem appears in [12], which confines itself to categories of sorted sets and does not characterise the equivalence as an object.

The abstract construction in this section does not require the variator Σ . Instead, we focus on the monad Σ° induced by its adjunction and generalise by fixing an arbitrary monad (T, η, μ) in \mathbf{C} . Let $F \dashv U: \mathbf{C} \rightleftarrows \mathcal{EM}(T)$ be the adjunction with its category of (Eilenberg-Moore) algebras. Given a \mathbf{C} -morphism $f: X \rightarrow UY$ for X in \mathbf{C} and Y in $\mathcal{EM}(T)$, we write $f^\sharp: FX \rightarrow Y$ for its adjoint transpose. We can then use a generalised version of the automata defined in Section 3.

► **Definition 5.1** (*T*-automaton). A *T*-automaton is a tuple (Q, δ, i, o) , where (Q, δ) is a *T*-algebra and $i: I \rightarrow Q$ and $o: Q \rightarrow O$ are morphisms in \mathcal{C} . A homomorphism from (Q, δ, i, o) to (Q', δ', i', o') is a *T*-algebra homomorphism $h: (Q, \delta) \rightarrow (Q', \delta')$ such that $h \circ i = i'$ and $o' \circ h = o$.

The reachability map of a *T*-automaton $\mathcal{A} = (Q, \delta, i, o)$ is given by $\text{rch}_{\mathcal{A}} = U(i^{\sharp}): TI \rightarrow Q$ and is therefore the unique *T*-algebra homomorphism $(TI, \mu) \rightarrow (Q, q)$ preserving initial states, taking $\eta_I: I \rightarrow TI$ to be the initial state selector of *TI*. The language of \mathcal{A} is given by $\mathcal{L}(\mathcal{A}) = o \circ \text{rch}_{\mathcal{A}}: TI \rightarrow O$.

The Σ -tree automata defined in Section 3 are recovered using the following fact: the category of Σ -algebras is isomorphic to $\mathcal{EM}(T)$ for *T* the free Σ -algebra monad Σ° .

► **Remark 5.2.** In **Set**, we may even add equations to the signature [36, Chapter VI.8, Theorem 1]. For **Nom**, this follows from the treatment of [34], giving a standard universal algebraic presentation of algebras over **Nom**. Indeed, results in this section apply to nominal tree automata, unless explicitly stated. For brevity we therefore focus on examples in **Set**.

► **Assumption 5.3.** In this section we will need the class \mathcal{E} to be the reflexive regular epis.³

The next lemma will be used in proving our main theorems.

► **Lemma 5.4.** Suppose *T* maps reflexive coequalisers to epimorphisms. If $i: B \rightarrow UC$ is such that $U(i^{\sharp})$ reflexively coequalises $q_1, q_2: A \rightarrow TB$ in \mathcal{C} , then i^{\sharp} reflexively coequalises $q_1^{\sharp}, q_2^{\sharp}: FA \rightarrow FB$.

Before defining an abstract Nerode equivalence, we recall the classical definition for languages of words. Given a language $L: A^* \rightarrow 2$, the equivalence $R \subseteq A^* \times A^*$ is defined as

$$R = \{(u, v) \in A^* \times A^* \mid \forall w \in A^*. L(uw) = L(vw)\}.$$

In this setting, $I = 1$ and $O = 2$. A function $Q \times A \rightarrow Q$ corresponds to an algebra for the monad $T = (-) \times A^*$, whose unit and multiplication are defined using the unit and multiplication of the monoid A^* . If $p_1, p_2: R \rightarrow A^* \cong 1 \times A^*$ are the projections, we note that R is defined to be the largest relation making the following diagram commute.

$$\begin{array}{ccc} R \times A^* & \xrightarrow{p_2 \times \text{id}} & 1 \times A^* \times A^* \\ \downarrow p_1 \times \text{id} & & \downarrow \mu \\ & & 1 \times A^* \\ & & \downarrow L \\ 1 \times A^* \times A^* & \xrightarrow{\mu} & 1 \times A^* \xrightarrow{L} 2 \end{array}$$

This leads to an abstract definition, using a limit⁴ to generalise what it means to be maximal.

► **Definition 5.5** (Nerode equivalence). Given a language $L: TI \rightarrow O$ and an object R with morphisms $p_1, p_2: R \rightarrow TI$, we say that (R, p_1, p_2) is the Nerode equivalence of L if the diagram below on the left commutes and for all objects S with a reflexive pair $q_1, q_2: S \rightarrow TI$

³ In a regular category, (reflexive regular epi, mono) forms a factorisation system in the same way (regular epi, mono) does, though one should note that the theory in this section does not actually need a factorisation system; the instantiation of \mathcal{E} is only invoked to obtain the right notion of reachability.

⁴ Note that it is not exactly a limit, as the defining property works with cones under *T*.

6:10 Tree Automata as Algebras: Minimisation and Determinisation

such that the diagram in the middle commutes there is a unique morphism $u: S \rightarrow R$ making the diagram on the right commute.

$$\begin{array}{ccc}
 TR & \xrightarrow{T p_2} & TTI \\
 \downarrow T p_1 & & \downarrow \mu \\
 TTI & \xrightarrow{\mu} & TI \\
 & \xrightarrow{L} & O
 \end{array}
 \quad
 \begin{array}{ccc}
 TS & \xrightarrow{T q_2} & TTI \\
 \downarrow T q_1 & & \downarrow \mu \\
 TTI & \xrightarrow{\mu} & TI \\
 & \xrightarrow{L} & O
 \end{array}
 \quad
 \begin{array}{ccc}
 & S & \\
 q_1 \swarrow & \downarrow u & \searrow q_2 \\
 TI & \xleftarrow{p_1} R \xrightarrow{p_2} & TI
 \end{array}$$

To show the versatility of our definition, we briefly explain a different example where the language is a set of words. This example cannot be recovered from the original definition by Arbib and Manes [8].

► **Example 5.6** (Syntactic congruence). Let T be the free monoid or list monad $(-)^*$ so that $\mathcal{EM}(T)$ is the category of monoids, $I = A$, and $O = 2$. Given a language $L: A^* \rightarrow 2$, the Nerode equivalence as defined above is then the largest relation $R \subseteq A^* \times A^*$ such that

$$\frac{n \in \mathbb{N} \quad (u_1, v_1), \dots, (u_n, v_n) \in R}{L(u_1 \cdots u_n) = L(v_1 \cdots v_n)}.$$

Equivalently, R is the largest relation such that

$$\frac{(u, v) \in R \quad w, x \in A^*}{L(wux) = L(wvx)},$$

which is precisely the *syntactic congruence* of the language.

We can show that the Nerode equivalence in **Set** exists, as long as the monad is finitary. To define it concretely, we use the following piece of notation. For any set X and $x \in X$, denote by $1_x: 1 \rightarrow X$ the constant x function, assuming no ambiguity of the set involved.

► **Proposition 5.7.** For $\mathbf{C} = \mathbf{Set}$ and T any finitary monad, every language $L: TI \rightarrow O$ has a Nerode equivalence given by

$$R = \{(u, v) \in TI \times TI \mid L \circ \mu \circ T[\text{id}_{TI}, 1_u] = L \circ \mu \circ T[\text{id}_{TI}, 1_v]: T(TI + 1) \rightarrow O\}$$

with the corresponding projections $p_1, p_2: R \rightarrow TI$.

The definition of R above states that $u, v \in TI$ are related iff the elements of TI formed by putting either u or v in any *context* and then applying μ have the same value under L . A context is an element of $T(TI + 1)$, where $1 = \{\square\}$ denotes a *hole* where either u or v can be plugged in. In the tree automata literature, such contexts, although restricted to contain a single instance of \square , are used in algorithms for minimisation [25] and learning [44, 19]. Unfortunately, the characterisation of Proposition 5.7 does not directly extend to **Nom**, because the functions 1_x are not, in general, equivariant. We leave this for future work.

Below we show that, under a few mild assumptions, the abstract equivalence is in fact a congruence: it induces a T -automaton, which moreover is minimal. Intuitively, given a language $L: TI \rightarrow O$ that has a Nerode equivalence, we use the equivalence to quotient the T -automaton (FI, η, L) . We first need a technical lemma.

► **Lemma 5.8.** If \mathbf{C} has coproducts, then for any Nerode equivalence (R, p_1, p_2) there exists a unique T -algebra structure $u: TR \rightarrow R$ making p_1 and p_2 T -algebra homomorphisms $(R, u) \rightarrow (TI, \mu)$ that have a common section.

► **Theorem 5.9.** *If \mathcal{C} has coproducts and reflexive coequalisers and T preserves reflexive coequalisers, then for every language that has a Nerode equivalence there exists a minimal T -automaton accepting it.*

Proof. Let $L: TI \rightarrow O$ be the language with Nerode equivalence (R, p_1, p_2) and $c: T \rightarrow M$ the coequaliser of p_1 and p_2 in \mathcal{C} . By Lemma 5.8 there exists a T -algebra structure on R making p_1 and p_2 T -algebra homomorphisms into (TI, μ) that have a common section. Since T preserves reflexive coequalisers, they are lifted by U , and we have a morphism $m: TM \rightarrow M$ making (M, m) a T -algebra such that c is a T -algebra homomorphism $(TI, \mu) \rightarrow (M, m)$. Since the diagram below on the left commutes and c coequalises p_1 and p_2 , there is a unique morphism o_M rendering the diagram on the right commutative.

$$\begin{array}{ccc}
 R & \xrightarrow{p_2} & TI \\
 \eta \searrow & \textcircled{1} & \downarrow \eta \\
 p_1 \downarrow & TR & \xrightarrow{Tp_2} TTI \xrightarrow{\mu} TI \\
 \textcircled{1} & \downarrow Tp_1 & \textcircled{2} \\
 TI & \xrightarrow{\eta} & TTI \\
 \textcircled{2} & \downarrow \mu & \textcircled{3} \\
 & TI & \xrightarrow{L} O
 \end{array}
 \quad
 \begin{array}{l}
 \textcircled{1} \text{ naturality of } \eta \\
 \textcircled{2} \text{ monad law} \\
 \textcircled{3} \text{ Nerode equivalence}
 \end{array}
 \quad
 \begin{array}{ccc}
 TI & \xrightarrow{c} & M \\
 L \searrow & & \downarrow o_M \\
 & & O
 \end{array}
 \quad (1)$$

Choosing $i_M = c \circ \eta: I \rightarrow M$, we obtain a T -automaton $\mathcal{M} = (M, m, i_M, o_M)$. Note that $U(i_M^\sharp) = c$, so c is the reachability map of \mathcal{M} . Hence, we find that $\mathcal{L}(\mathcal{M}) = L$ by (1). The morphism c coequalises the reflexive pair (p_1, p_2) by definition, so \mathcal{M} is reachable.

To see that \mathcal{M} is minimal, consider any reachable T -automaton $\mathcal{A} = (Q, \delta, i, o)$ such that $\mathcal{L}(\mathcal{A}) = L$. Reachability amounts to the reachability map $\text{rch}: TI \rightarrow UQ$ being the reflexive coequaliser of a pair of morphisms $q_1, q_2: S \rightarrow TI$. From commutativity of

$$\begin{array}{ccc}
 TS & \xrightarrow{Tq_2} & TTI \\
 \textcircled{1} & \searrow T\text{rch} & \downarrow \mu \\
 TQ & & TI \\
 Tq_1 \downarrow & \textcircled{2} & \downarrow L \\
 TTI & \xrightarrow{\mu} & TI \\
 \textcircled{3} & & \textcircled{3}
 \end{array}
 \quad
 \begin{array}{l}
 \textcircled{1} \text{ rch coequalises } q_1 \text{ and } q_2 \\
 \textcircled{2} \text{ rch is a } T\text{-algebra homomorphism} \\
 \textcircled{3} \mathcal{L}(\mathcal{A}) = L
 \end{array}$$

we obtain by the Nerode equivalence property a unique morphism $v: S \rightarrow R$ making the diagram below on the left commute.

$$\begin{array}{ccc}
 S & & \\
 q_1 \swarrow & \downarrow v & \searrow q_2 \\
 TI & \xleftarrow{p_1} R & \xrightarrow{p_2} TI
 \end{array}
 \quad
 \begin{array}{ccc}
 S & \xrightarrow{q_2} & TI \\
 q_1 \downarrow & \searrow v & \downarrow c \\
 TI & \xleftarrow{p_1} R & \xrightarrow{p_2} TI \\
 & \swarrow c & \downarrow c \\
 & & M
 \end{array}$$

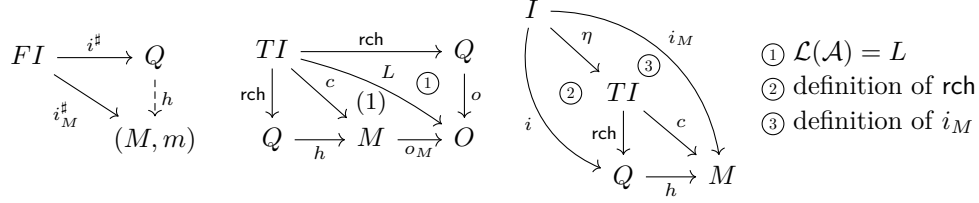
Extending this with c , the coequaliser of p_1 and p_2 , gives the commutative diagram on the right. Recall that $U(i_M^\sharp) = c$. We now find

$$i_M^\sharp \circ q_1^\sharp = (U(i_M^\sharp) \circ q_1)^\sharp = (c \circ q_1)^\sharp = (c \circ q_2)^\sharp = (U(i_M^\sharp) \circ q_2)^\sharp = i_M^\sharp \circ q_2^\sharp.$$

Here the first and last equality apply a general naturality property of the adjunction. Since $\text{rch} = U(i^\sharp)$ is the reflexive coequaliser of q_1 and q_2 , i^\sharp is the reflexive coequaliser of q_1^\sharp and

6:12 Tree Automata as Algebras: Minimisation and Determinisation

q_2^\sharp by Lemma 5.4. We then obtain a unique T -algebra homomorphism $h: (Q, \delta) \rightarrow (M, m)$ making the diagram below on the left commute.



From commutativity of the other diagrams we find $o_M \circ h = o$ (using that rch is epi) and $h \circ i = i_M$. Thus, h is a T -automaton homomorphism $\mathcal{A} \rightarrow \mathcal{M}$. To see that it is unique, note that any T -automaton homomorphism $h': \mathcal{A} \rightarrow \mathcal{M}$ is a T -algebra homomorphism $(Q, \delta) \rightarrow (M, m)$ such that $h' \circ i = i_M$. It is then not hard to see that $h' \circ i^\sharp = (h' \circ i)^\sharp = i_M^\sharp$. We conclude that $h' = h$ by the uniqueness property of h satisfying $h \circ i^\sharp = i_M^\sharp$. \blacktriangleleft

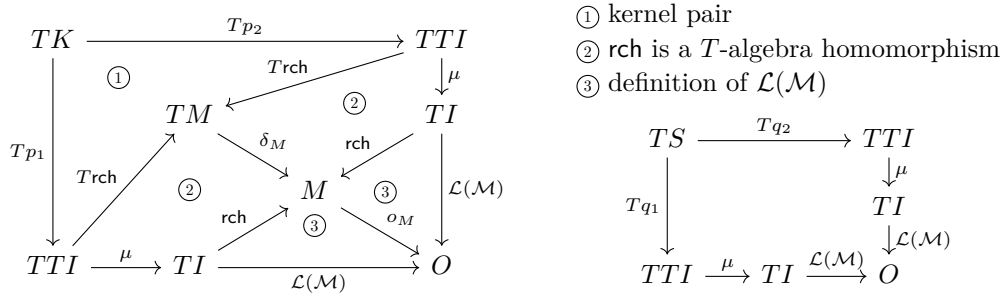
► **Remark 5.10.** We briefly discuss the conditions of the above theorem in the specific case of $\mathbf{C} = \mathbf{Set}$ with T a finitary monad. This includes the setting of tree automata in \mathbf{Set} , as a monad on \mathbf{Set} is finitary if and only if $\mathcal{EM}(T)$ is equivalent to the category of algebras for a signature modulo equations. Proposition 5.7 shows that all Nerode equivalences exist here. Furthermore, Lack and Rosický [35] observe that an endofunctor on \mathbf{Set} is finitary if and only if it preserves sifted colimits, of which reflexive coequalisers form an instance.

To conclude this section we show that the converse of the previous theorem also holds, using the existence of kernel pairs rather than coproducts. We need a technical lemma first.

► **Lemma 5.11.** *If $q_1, q_2: A \rightarrow TB$ is a reflexive pair in \mathbf{C} , then so is (q_1^\sharp, q_2^\sharp) in $\mathcal{EM}(T)$.*

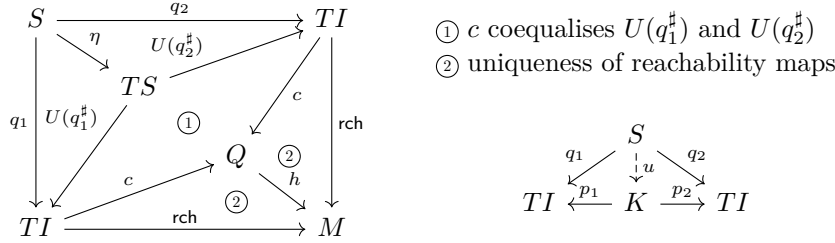
► **Theorem 5.12.** *If \mathbf{C} has kernel pairs and reflexive coequalisers and T preserves reflexive coequalisers, then every language that has a minimal T -automaton has a Nerode equivalence.*

Proof. Let $\mathcal{M} = (M, \delta_M, i_M, o_M)$ be a minimal T -automaton and $p_1, p_2: K \rightarrow TI$ the kernel pair of its reachability map $\text{rch}: FI \rightarrow M$. We claim that K together with p_1 and p_2 forms the Nerode equivalence of $\mathcal{L}(\mathcal{M})$. To see this, note that the diagram below on the left commutes.



Now if S with $q_1, q_2: S \rightarrow TI$ is any reflexive pair making the diagram on the right commute, we let $c: TI \rightarrow Q$ be the coequaliser of $U(q_1^\sharp)$ and $U(q_2^\sharp)$, noting that this is a reflexive pair by Lemma 5.11. Then since T preserves reflexive coequalisers, they are lifted by U , meaning that there exists a unique T -algebra structure $\delta: TQ \rightarrow Q$ making $c: FI \rightarrow (Q, \delta)$ a T -algebra homomorphism that is the coequaliser of q_1^\sharp and q_2^\sharp . We also have $\mathcal{L}(\mathcal{M}) \circ U(q_1^\sharp) = \mathcal{L}(\mathcal{M}) \circ U(q_2^\sharp)$ by commutativity of the diagram on the right, so with c coequalising $U(q_1^\sharp)$ and $U(q_2^\sharp)$ there is a unique morphism $o: Q \rightarrow O$ such that $o \circ c = \mathcal{L}(\mathcal{M})$. Setting $i = c \circ \eta_I$, we have a

T -automaton (Q, δ, i, o) with reachability map $U(i^\sharp) = c$ that accepts the language $\mathcal{L}(\mathcal{M})$. By \mathcal{M} being minimal there exists a unique T -automaton homomorphism $h: (Q, \delta, i, o) \rightarrow \mathcal{M}$. Then the diagram below on the left commutes.



By p_1 and p_2 being the kernel pair of rch there exists a unique morphism $u: S \rightarrow K$ making the diagram on the right commute. ◀

6 Tree automata with side-effects

We extend tree automata with various side-effects, covering as examples non-deterministic, weighted, and non-deterministic nominal automata. The key insight is to view them as algebras in the *Kleisli category of a monad* S . We first recall some basic notions.

Kleisli category. Every monad (S, η, μ) has an associated *Kleisli category* $\mathcal{Kl}(S)$, whose objects are those of \mathbf{C} and whose morphisms $X \rightarrowtail Y$ are morphisms $X \rightarrow SX$ in \mathbf{C} . Given such morphisms $f: X \rightarrowtail Y$ and $g: Y \rightarrowtail Z$, their (Kleisli) composition $g \circ f$ is defined as $\mu_Z \circ Sg \circ f$ in \mathbf{C} . The *Kleisli adjunction* $J \dashv V: \mathbf{C} \rightleftarrows \mathcal{Kl}(S)$ is given by $JX = X$, $J(f: X \rightarrow Y) = \eta_Y \circ f$ and $VY = SY$, $V(f: X \rightarrowtail Y) = \mu_Y \circ Sf$.

► **Example 6.1.** The category $\mathcal{Kl}(\mathcal{P}_f)$ has morphisms $X \rightarrow \mathcal{P}_f Y$, which are finitely-branching relations, and Kleisli composition is relational composition. We have that J maps a function to its graph, and V maps X to $\mathcal{P}_f X$ and a relation $R: X \rightarrowtail Y$ to the function

$$\lambda U \subseteq X.\{y \mid \exists x \in U : (x, y) \in R\}.$$

The category $\mathcal{Kl}(\mathcal{M}_{\mathbb{F}})$ has morphisms $X \rightarrow \mathcal{M}_{\mathbb{F}} Y$ that are matrices over \mathbb{F} indexed by X and Y (equivalently, linear maps between the corresponding free vector spaces), and composition is matrix multiplication. The left adjoint J maps a function $f: X \rightarrow Y$ to the matrix $Jf[x, f(x)] = 1$, for $x \in X$, and 0 elsewhere, and the right adjoint V maps a matrix $X \rightarrowtail Y$ to the corresponding linear function $\mathcal{M}_{\mathbb{F}} X \rightarrow \mathcal{M}_{\mathbb{F}} Y$.

Given an endofunctor Σ on \mathbf{C} , a functor $\widehat{\Sigma}: \mathcal{Kl}(S) \rightarrow \mathcal{Kl}(S)$ is an *extension* of Σ if the following diagram commutes:

$$\begin{array}{ccc} \mathbf{C} & \xrightarrow{\Sigma} & \mathbf{C} \\ J \downarrow & & \downarrow J \\ \mathcal{Kl}(S) & \xrightarrow{\widehat{\Sigma}} & \mathcal{Kl}(S) \end{array}$$

Extensions are in bijective correspondence with *distributive laws* $\lambda: \Sigma S \Rightarrow S\Sigma$ [39], which are natural transformations satisfying certain axioms. Explicitly, we have $\widehat{\Sigma}X = \Sigma X$ and $f: X \rightarrowtail Y$ (a morphism $X \rightarrow SY$ in \mathbf{C}) is mapped to $\lambda_Y \circ \Sigma f: \Sigma X \rightarrow S\Sigma Y$, seen in $\mathcal{Kl}(S)$.

In [23, Lemma 2.4] it is shown that a canonical distributive law in \mathbf{Set} always exists in case Σ is polynomial and S is a commutative monad [31].

► **Example 6.2.** For Σ a polynomial Set endofunctor, the canonical distributive law $\lambda: \Sigma \mathcal{P}_f \Rightarrow \mathcal{P}_f \Sigma$ can be directly defined as follows: $\lambda_X(u) = \{v \in \Sigma X \mid (v, u) \in \text{img}(\langle \Sigma p_1, \Sigma p_2 \rangle)\}$, where p_1 and p_2 are the left and right projections of the membership relation $\in_X \subseteq X \times \mathcal{P}_f X$.

The multiplicity monad $(\mathcal{M}_{\mathbb{F}}, e, m)$ admits a distributive law $\lambda: \Sigma \mathcal{M}_{\mathbb{F}} \Rightarrow \mathcal{M}_{\mathbb{F}} \Sigma$, inductively defined as follows, where \otimes is the Kronecker product:

$$\lambda^{\text{id}} = \text{id}_{\mathcal{M}_{\mathbb{F}}} \quad \lambda^A = e_A \quad \lambda^{\Sigma_1 \times \Sigma_2} = \otimes \circ (\lambda^{\Sigma_1} \times \lambda^{\Sigma_2}) \quad \lambda^{\prod_{i \in I} \Sigma_i} = [\mathcal{M}_{\mathbb{F}}(\kappa_i) \circ \lambda^{\Sigma_i}]_{i \in I}$$

We can now define our notion of tree automaton with side-effects.

► **Definition 6.3** ((Σ, S)-tree automaton). *Given a monad S , and $\widehat{\Sigma}: \mathcal{Kl}(S) \rightarrow \mathcal{Kl}(S)$ extending a functor Σ on \mathbf{C} , a (Σ, S)-tree automaton is a $\widehat{\Sigma}$ -tree automaton, i.e., a tuple (Q, δ, i, o) , where (Q, δ) is a $\widehat{\Sigma}$ -algebra and $i: I \rightarrow Q$ and $o: Q \rightarrow O$ are morphisms in $\mathcal{Kl}(S)$.*

► **Example 6.4.**

1. Let Γ be a signature, and let Σ be its signature functor. Then the extension of Σ to $\mathcal{Kl}(\mathcal{P}_f)$ is obtained via the distributive law of Example 6.2, namely $\widehat{\Sigma}X = \Sigma X$ and

$$\widehat{\Sigma}(f: X \rightarrow Y)(\kappa_\gamma(x_1, \dots, x_{\text{ar}(\gamma)})) = \{(\kappa_\gamma(y_1, \dots, y_{\text{ar}(\gamma)})) \mid y_j \in f(x_j) \text{ for } 1 \leq j \leq \text{ar}(\gamma)\}$$

for $\gamma \in \Gamma$. Non-deterministic tree automata are (Σ, \mathcal{P}) -tree automata (Q, δ, i, o) with $O = 1$, the singleton set. In fact, we have that $\delta: \prod_{\gamma \in \Gamma} Q^{\text{ar}(\gamma)} \rightarrow Q$ is a family of relations $\delta_\gamma \subseteq Q^{\text{ar}(\gamma)} \times Q$; by the same token, $i \subseteq I \times Q$ relates the frontier alphabet with (possibly several) states, and $o \subseteq Q \times 1 \cong Q$ is the set of final states.

2. Let Γ be a signature, and let Σ be its signature functor. The extension of Σ to $\mathcal{Kl}(\mathcal{M}_{\mathbb{F}})$ is obtained via the distributive law of Example 6.2. Explicitly, $\widehat{\Sigma}X = \Sigma X$ and $\widehat{\Sigma}(f: X \rightarrow Y)$ maps $\kappa_\gamma(x_1, \dots, x_{\text{ar}(\gamma)})$ to the vector $[\kappa_\gamma(v_z)]_{z \in Y^{\text{ar}(\gamma)}}$ such that v_z is the z -th component of $f(x_1) \otimes \dots \otimes f(x_{\text{ar}(\gamma)})$, for $\gamma \in \Gamma$. A multiplicity tree automaton [29] is a $(\Sigma, \mathcal{M}_{\mathbb{F}})$ -tree automaton (Q, δ, i, o) with $I = O = 1$. In fact, we have that δ_γ is the *transition matrix* $Q^{\text{ar}(\gamma)} \rightarrow Q$ in $\mathbb{F}^{|Q|^{\text{ar}(\gamma)} \times |Q|}$; similarly, $i: 1 \rightarrow Q$ is the *initial weight vector* in $\mathbb{F}^{1 \times |Q|}$, and $o: Q \rightarrow 1$ is the *final weight vector* in $\mathbb{F}^{|Q| \times 1}$. Intuitively, δ_γ maps an $\text{ar}(\gamma)$ -tuple of elements of Q to a linear combination over Q . We note that we can go beyond fields and consider $(\Sigma, \mathcal{M}_{\mathbb{S}})$ -tree automata for a semiring \mathbb{S} , encompassing *weighted* tree automata [17].
3. The nondeterministic version of the automata defined in Section 3.1 can be obtained via the monad $\mathcal{P}_\omega: \mathbf{Nom} \rightarrow \mathbf{Nom}$, mapping a nominal set to the nominal set of its finitely-supported, orbit-finite subsets.⁵ This is analogous to non-deterministic nominal automata [13]. We note that $\mathcal{Kl}(\mathcal{P}_\omega)$ is precisely the category of nominal sets and (orbit-finitely branching) equivariant relations and that Σ_λ extends to relations just as a set endofunctor – the distributive law is defined as the one for \mathcal{P}_f of Example 6.2, where all the maps are equivariant. A nondeterministic nominal Σ_λ -tree automaton is a $(\Sigma_\lambda, \mathcal{P}_\omega)$ -tree automaton (Q, δ, i, o) with $O = 1$, the one-element nominal set with trivial group action. We have that i and the components of δ are equivariant relations. For instance, $\delta_{\text{lambda}} \subseteq [\mathbb{A}]Q \times Q$, and o is an equivariant subset of Q .

We now study language semantics of (Σ, S) -tree automata. Languages are defined via free algebras (see Section 3). It turns out that the free algebras in $\text{Alg}(\Sigma)$ and $\text{Alg}(\widehat{\Sigma})$ are closely related. To see this, we use the following result, which follows from [24, Theorem 2.14].

⁵ This is the finitary version of the powerset functor in \mathbf{Nom} .

► **Lemma 6.5.** *Let $\Sigma: \mathbf{C} \rightarrow \mathbf{C}$ be a functor, $S: \mathbf{C} \rightarrow \mathbf{C}$ a monad, and $\widehat{\Sigma}: \mathcal{Kl}(S) \rightarrow \mathcal{Kl}(S)$ an extension of Σ . Let $\lambda: \Sigma S \Rightarrow S\Sigma$ be the corresponding distributive law.*

Then the Kleisli adjunction $J \dashv V: \mathbf{C} \rightleftarrows \mathcal{Kl}(S)$ lifts to an adjunction in: $\overline{J} \dashv \overline{V}$

$$\begin{array}{ccc}
 \text{Alg}(\Sigma) & \xrightarrow{\overline{J}} & \text{Alg}(\widehat{\Sigma}) \\
 \downarrow & \perp & \downarrow \\
 \mathbf{C} & \xrightarrow{J} & \mathcal{Kl}(S) \\
 \downarrow & \perp & \downarrow \\
 \mathbf{C} & \xrightarrow{V} & \mathcal{Kl}(S)
 \end{array}
 \quad
 \begin{array}{l}
 \overline{J}(Q, \delta: \Sigma Q \rightarrow Q) = (Q, J(\delta): \widehat{\Sigma}Q \rightarrow Q) \\
 \overline{V}(Q, \gamma: \widehat{\Sigma}Q \rightarrow Q) = (SQ, V(\gamma) \circ \lambda_Q: \Sigma SQ \rightarrow SQ)
 \end{array}$$

From Lemma 6.5, and using that free algebras can be obtained as colimits of transfinite sequences [2, 28], it follows that any free Σ -algebra $(\Sigma^\circ X, \alpha_X)$ is mapped by the left adjoint \overline{J} to a free $\widehat{\Sigma}$ -algebra with the same carrier. Concretely, given a $\widehat{\Sigma}$ -algebra (Q, δ) and a morphism $i: I \rightarrow Q$, a (unique) morphism $\text{rch}: \Sigma^\circ I \rightarrow SQ$ makes the diagram on the left commute in \mathbf{C} iff it makes the diagram on the right commute in $\mathcal{Kl}(S)$:

$$\begin{array}{ccc}
 \Sigma\Sigma^\circ I & \xrightarrow{\alpha} & \Sigma^\circ I \xleftarrow{\eta} I \\
 \Sigma\text{rch} \downarrow & & \text{rch} \downarrow \swarrow i \\
 \Sigma SQ & \xrightarrow{\overline{V}(Q, \delta)} & SQ
 \end{array}
 \quad
 \begin{array}{ccc}
 \widehat{\Sigma}\Sigma^\circ I & \xrightarrow{\overline{J}(\Sigma^\circ I, \alpha)} & \Sigma^\circ I \xleftarrow{J\eta} I \\
 \widehat{\Sigma}\text{rch} \downarrow & & \text{rch} \downarrow \swarrow i \\
 \widehat{\Sigma}Q & \xrightarrow{\delta} & Q
 \end{array}
 \quad (2)$$

Note that the adjoint transpose of rch is the same morphism, seen in $\mathcal{Kl}(S)$. The functor \overline{V} can be viewed as a *determinisation* construction for (Σ, S) -tree automata.

► **Definition 6.6.** *Given an (Σ, S) -tree automaton (Q, δ, i, o) , let $\bar{\delta}: \Sigma S(Q) \rightarrow S(Q)$ be the algebra structure of $\overline{V}(Q, \delta)$, i.e., $(S(Q), \bar{\delta}) = \overline{V}(Q, \delta)$, and $\bar{o} = V(o)$. The Σ -tree automaton $(SQ, \bar{\delta}, i, \bar{o})$ is called the *determinisation* of (Q, δ, i, o) .*

The following shows correctness of this determinisation construction, using the correspondence in (2), and provides a concrete description of the language semantics of (Σ, S) -tree automata.

► **Corollary 6.7.** *Let (Q, δ, i, o) be a (Σ, S) -tree automaton. Then $\mathcal{L}(SQ, \bar{\delta}, i, \bar{o}) = \mathcal{L}(Q, \delta, i, o)$.*

We conclude this section with some example instantiations of determinisation, and of how they can be used to compute languages.

► **Example 6.8.**

1. The determinisation of a (Σ, \mathcal{P}_f) -tree automaton (Q, δ, i, o) is

$$\bar{\delta}_\gamma(X_1, \dots, X_k) = \bigcup_{x_1 \in X_1, \dots, x_k \in X_k} \delta_\gamma(x_1, \dots, x_k) \quad \bar{o}(X) = \bigcup_{x \in X} o(x)$$

for $\gamma \in \Gamma$ with $k = \text{ar}(\gamma)$, and X_1, \dots, X_k, X finite subsets of Q . This definition precisely corresponds to the usual determinisation of bottom-up tree automata (see e.g. [22]). The reachability function is then given by

$$\text{rch}(t) = \begin{cases} i(t) & \text{if } t \in I \\ \bigcup_{\substack{x_1 \in \text{rch}(t_1) \\ \dots \\ x_k \in \text{rch}(t_k)}} \delta_\gamma(x_1, \dots, x_k) & \text{if } t = (\gamma, x_1, \dots, x_k), k = \text{ar}(\gamma). \end{cases}$$

Using Corollary 6.7, we have that the language of (Q, δ, i, o) is

$$\mathcal{L}(Q, \delta, i, o)(t) = \bigcup_{s \in \text{rch}(t)} o(s).$$

That is: a tree t is accepted by (Q, δ, i, o) whenever there is a final state among those reached by parsing t .

2. The determinisation of a $(\Sigma, \mathcal{M}_{\mathbb{F}})$ -tree automaton is given by

$$\bar{\delta}_{\gamma}(\varphi_1, \dots, \varphi_k) = (\varphi_1 \otimes \dots \otimes \varphi_k) \bullet \delta_{\gamma} \quad \bar{o}(\varphi) = \varphi \bullet o$$

where $\gamma \in \Gamma$ and $\text{ar}(\gamma) = k$; also, \otimes is the Kronecker product and \bullet is matrix multiplication. Explicitly, $\bar{\delta}_{\gamma}$ takes a k -tuple of vectors over Q and turns it into a vector φ over k -tuples of states via the distributive law (defined as the Kronecker product, see Example 6.2), i.e., $\varphi(q_1, \dots, q_k) = \varphi_1(q_1) \dots \varphi_k(q_k)$, for $q_1, \dots, q_k \in Q$. The result is then multiplied by the matrix δ_{γ} to compute the successor vector.

Similarly, the reachability function becomes

$$\text{rch}(t) = \begin{cases} i(t) & \text{if } t \in I \\ (\text{rch}(t_1) \otimes \dots \otimes \text{rch}(t_k)) \bullet \delta_{\gamma} & \text{if } t = (\gamma, t_1, \dots, t_k), k = \text{ar}(\gamma). \end{cases}$$

Hence we obtain the language $\mathcal{L}(Q, \delta, i, o)(t) = \text{rch}(t) \bullet o$, which corresponds to the language semantics given in [29].

3. The case of $(\Sigma_{\lambda}, \mathcal{P}_{\omega})$ -tree automata is completely analogous to point 1. For instance,

$$\bar{\delta}_{\text{lambd}}(X) = \bigcup_{x \in X} \delta_{\text{lambd}}(x)$$

for X a finitely supported orbit-finite subset of $[\mathbb{A}]Q$.

7 Future work

The algorithmic side of the iterative minimisation construction presented in Section 4 is left open. For classical tree automata there exist sophisticated variants of partition refinement [25, 1], akin to Hopcroft's classical algorithm. A generalisation to the current algebraic setting is an interesting direction of research, for which a natural starting point would be to try and integrate in our setting the efficient coalgebraic algorithm presented in [18].

Further, we characterised the minimal automaton as the greatest fixed point of a monotone function, recovering the notion of forward bisimulations as its post-fixed points (although it is perhaps more natural to think of these as congruences). This characterisation suggests an integration with up-to techniques [42, 14, 15], which have, to the best of our knowledge, not been applied to tree automata. In particular, we are interested in applying these algorithms to decide equivalence of series-parallel rational and series-rational expressions [38].

Since completeness of Kleene Algebra is connected to minimality of deterministic finite automata [33], we wonder whether a completeness proof can be recovered using automata as presented in this paper. In particular, our abstract framework might allow us to transpose such a proof to settings such as Bi-Kleene Algebra [37] or Concurrent Kleene Algebra [27].

References

- 1 Parosh Aziz Abdulla, Johanna Högberg, and Lisa Kaati. Bisimulation Minimization of Tree Automata. *Int. J. Found. Comput. Sci.*, 18(4):699–713, 2007. doi:10.1142/S0129054107004929.
- 2 Jiří Adámek. Free algebras and automata realizations in the language of categories. *Commentationes Mathematicae Universitatis Carolinae*, 15(4):589–602, 1974. URL: <http://eudml.org/doc/16649>.

- 3 Jiří Adámek. Realization theory for automata in categories. *J. Pure and Appl. Algebra*, 9(2):281–296, 1977. doi:10.1016/0022-4049(77)90071-8.
- 4 Jiří Adámek, Filippo Bonchi, Mathias Hülsbusch, Barbara König, Stefan Milius, and Alexandra Silva. A Coalgebraic Perspective on Minimization and Determinization. In *FoSSaCS*, pages 58–73, 2012. doi:10.1007/978-3-642-28729-9_4.
- 5 Jiří Adámek, Horst Herrlich, and George Strecker. *Abstract and concrete categories: The joy of cats*, volume 17 of *Reprints in Theory and Applications of Categories*. TAC, 2006.
- 6 Jiří Adámek and Vera Trnková. *Automata and algebras in categories*. Kluwer, 1989.
- 7 Brian D.O. Anderson, Michael A. Arbib, and Ernest G. Manes. *Foundations of system theory: finitary and infinitary conditions*, volume 115 of *Lecture Notes in Econ. and Math. Syst.* Springer, 1976.
- 8 Michael A. Arbib and Ernest G. Manes. Machines in a category: An expository introduction. *SIAM review*, 16(2):163–192, 1974.
- 9 Michael A. Arbib and Ernest G. Manes. Adjoint machines, state-behavior machines, and duality. *Journal of Pure and Applied Algebra*, 6(3):313–344, 1975.
- 10 Simone Barlocco, Clemens Kupke, and Jurriaan Rot. Coalgebra Learning via Duality. In *FoSSaCS*, pages 62–79, 2019. doi:10.1007/978-3-030-17127-8_4.
- 11 Alwin Blok. Interaction, observation and denotation. Master’s thesis, ILLC Amsterdam, 2012.
- 12 Mikołaj Bojańczyk. Recognisable languages over monads. In *DLT*, pages 1–13. Springer, 2015.
- 13 Mikołaj Bojańczyk, Bartek Klin, and Slawomir Lasota. Automata theory in nominal sets. *Logical Methods in Computer Science*, 10(3), 2014. doi:10.2168/LMCS-10(3:4)2014.
- 14 Filippo Bonchi, Daniela Petrisan, Damien Pous, and Jurriaan Rot. A general account of coinduction up-to. *Acta Inf.*, 54(2):127–190, 2017. doi:10.1007/s00236-016-0271-4.
- 15 Filippo Bonchi and Damien Pous. Hacking nondeterminism with induction and coinduction. *Commun. ACM*, 58(2):87–95, 2015. doi:10.1145/2713167.
- 16 Francis Borceux. *Handbook of Categorical Algebra*, volume 1 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 1994. doi:10.1017/CB09780511525858.
- 17 Björn Borchardt and Heiko Vogler. Determinization of Finite State Weighted Tree Automata. *Journal of Automata, Languages and Combinatorics*, 8(3):417–463, 2003.
- 18 Ulrich Dorsch, Stefan Milius, Lutz Schröder, and Thorsten Wißmann. Efficient Coalgebraic Partition Refinement. In *CONCUR*, pages 32:1–32:16, 2017. doi:10.4230/LIPIcs.CONCUR.2017.32.
- 19 Frank Drewes and Johanna Högberg. Query learning of regular tree languages: How to avoid dead states. *Theory of Computing Systems*, 40(2):163–185, 2007.
- 20 Marcelo P. Fiore. Discrete Generalised Polynomial Functors (Extended Abstract). In *ICALP*, pages 214–226, 2012.
- 21 Murdoch James Gabbay and Aad Mathijssen. Nominal (Universal) Algebra: Equational Logic with Names and Binding. *J. Log. Comput.*, 19(6):1455–1508, 2009. doi:10.1093/logcom/exp033.
- 22 Amaury Habrard and José Oncina. Learning Multiplicity Tree Automata. In *ICGI*, volume 4201 of *Lecture Notes in Computer Science*, pages 268–280. Springer, 2006.
- 23 Ichiro Hasuo, Bart Jacobs, and Ana Sokolova. Generic Trace Semantics via Coinduction. *Logical Methods in Computer Science*, 3(4), 2007. doi:10.2168/LMCS-3(4:11)2007.
- 24 Claudio Hermida and Bart Jacobs. Structural Induction and Coinduction in a Fibrational Setting. *Inf. Comput.*, 145(2):107–152, 1998. doi:10.1006/inco.1998.2725.
- 25 Johanna Högberg, Andreas Maletti, and Jonathan May. Backward and forward bisimulation minimization of tree automata. *Theoretical Computer Science*, 410(37):3539–3552, 2009. doi:10.1016/j.tcs.2009.03.022.
- 26 William M. Holcombe. *Algebraic Automata Theory*. Cambridge University Press, 1982.
- 27 Tobias Kappé, Paul Brunet, Alexandra Silva, and Fabio Zanasi. Concurrent Kleene Algebra: Free Model and Completeness. In *ESOP*, pages 856–882, 2018. doi:10.1007/978-3-319-89884-1_30.

- 28 Gregory M. Kelly. A unified treatment of transfinite constructions for free algebras, free monoids, colimits, associated sheaves, and so on. *Bull. Austr. Math. Soc.*, 22(1):1–83, 1980. doi:10.1017/S0004972700006353.
- 29 Stefan Kiefer, Ines Marusic, and James Worrell. Minimisation of Multiplicity Tree Automata. *Logical Methods in Computer Science*, 13(1), 2017.
- 30 Bartek Klin and Jurriaan Rot. Coalgebraic trace semantics via forgetful logics. *Logical Methods in Computer Science*, 12(4), 2016. doi:10.2168/LMCS-12(4:10)2016.
- 31 Anders Kock. Monads on symmetric monoidal closed categories. *Arch. der Math.*, 21(1):1–10, 1970. doi:10.1007/BF01220868.
- 32 Barbara König and Sebastian Küpper. Generic Partition Refinement Algorithms for Coalgebras and an Instantiation to Weighted Automata. In *Theory Comput. Syst.*, pages 311–325, 2014. doi:10.1007/978-3-662-44602-7_24.
- 33 Dexter Kozen. A Completeness Theorem for Kleene Algebras and the Algebra of Regular Events. *Inf. Comput.*, 110(2):366–390, 1994. doi:10.1006/inco.1994.1037.
- 34 Alexander Kurz and Daniela Petrisan. On universal algebra over nominal sets. *Mathematical Structures in Computer Science*, 20(2):285–318, 2010. doi:10.1017/S0960129509990399.
- 35 Stephen Lack and Jiří Rosický. Notions of Lawvere theory. *Appl. Categ. Struct.*, 19(1):363–391, 2011.
- 36 Saunders Mac Lane. *Categories for the working mathematician*, volume 5. Springer, 2013.
- 37 Michael R. Laurence and Georg Struth. Completeness Theorems for Bi-Kleene Algebras and Series-Parallel Rational Pomset Languages. In *RAMiCS*, pages 65–82, 2014. doi:10.1007/978-3-319-06251-8_5.
- 38 Kamal Lodaya and Pascal Weil. Series-parallel languages and the bounded-width property. *Theoretical Computer Science*, 237(1):347–380, 2000. doi:10.1016/S0304-3975(00)00031-1.
- 39 Philip S. Mulry. Lifting Theorems for Kleisli Categories. In *MFPS*, pages 304–319, 1993. doi:10.1007/3-540-58027-1_15.
- 40 Jean Éric Pin. *Mathematical Foundations of Automata Theory*. Version of March 13, 2019.
- 41 Andrew M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge University Press, 2013.
- 42 Damien Pous and Davide Sangiorgi. *Advanced Topics in Bisimulation and Coinduction*, chapter Enhancements of the coinductive proof method. Cambridge University Press, 2011.
- 43 Jan J. M. M. Rutten. Automata and Coinduction (An Exercise in Coalgebra). In *CONCUR*, pages 194–218, 1998. doi:10.1007/BFb0055624.
- 44 Yasubumi Sakakibara. Learning context-free grammars from structural data in polynomial time. *Theoretical Computer Science*, 76(2-3):223–242, 1990.
- 45 Alexandra Silva, Filippo Bonchi, Marcello M. Bonsangue, and Jan J. M. M. Rutten. Generalizing determinization from automata to coalgebras. *Logical Methods in Computer Science*, 9(1), 2013. doi:10.2168/LMCS-9(1:9)2013.
- 46 Thorsten Wißmann, Stefan Milius, Shin-ya Katsumata, and Jérémy Dubut. A Coalgebraic View on Reachability. *arXiv e-prints*, January 2019. arXiv:1901.10717.

A Proofs for Section 5

In the proofs below, we will use the following basic adjunction properties, in particular for the adjunction $F \dashv U: \mathbf{C} \rightleftharpoons \mathcal{EM}(T)$ with adjoint transpose $(-)^{\sharp}$:

- The transpose $f^{\sharp}: FA \rightarrow B$ for $f: A \rightarrow UB$ in \mathbf{C} can be defined as $f^{\sharp} = y \circ Tf$, where y is the T -algebra structure on Y .
- For all $f: X \rightarrow UY$ and $g: UY \rightarrow UZ$ in \mathbf{C} we have $U(g^{\sharp}) \circ Tf = U((g \circ f)^{\sharp})$.
- We have $U(\eta_X^{\sharp}) = \text{id}_{TX}$.

We need the following additional lemmas in the proofs below.

► **Lemma A.1.** *If $f: A \rightarrow B$ and $h: A \rightarrow C$ in $\mathcal{EM}(T)$ are such that there exists $g: UB \rightarrow UC$ in \mathcal{C} with $g \circ f = h$ and Tf is an epi, then g is a T -algebra homomorphism $B \rightarrow C$.*

Proof. Let $\alpha: TA \rightarrow A$, $\beta: TB \rightarrow B$, and $\gamma: TC \rightarrow C$ be the respective T -algebra structures on A , B , and C . By commutativity of

$$\begin{array}{ccccc}
 TA & \xrightarrow{Tf} & TB & & \\
 \downarrow Tf & \searrow \alpha & \searrow Th & \searrow Tg & \\
 & A & & TC & \\
 & \downarrow f & & \downarrow \gamma & \\
 TB & \xrightarrow{\beta} & B & \xrightarrow{g} & C
 \end{array}$$

and Tf being an epi, we directly conclude that g is a T -algebra homomorphism $B \rightarrow C$. ◀

► **Lemma A.2.** *Given a language L and $q_1, q_2: S \rightarrow TI$ making the diagram below on the left commute, the diagram on the right commutes.*

$$\begin{array}{ccc}
 TS & \xrightarrow{Tq_2} & TTI \\
 \downarrow Tq_1 & & \downarrow \mu \\
 TTI & \xrightarrow{\mu} & TI \xrightarrow{L} O
 \end{array}
 \qquad
 \begin{array}{ccc}
 TTS & \xrightarrow{TTq_2} & TTTI \xrightarrow{T\mu} TTI \\
 TTq_1 \downarrow & & \downarrow \mu \\
 TTTI & & TI \\
 T\mu \downarrow & & \downarrow L \\
 TTI & \xrightarrow{\mu} & TI \xrightarrow{L} O
 \end{array}$$

Furthermore, if (q_1, q_2) is a reflexive pair, then so is $(\mu_I \circ Tq_1, \mu_I \circ Tq_2)$.

Proof. We extend the assumption to the following commutative diagram.

$$\begin{array}{ccccc}
 TTS & \xrightarrow{TTq_2} & TTTI & \xrightarrow{T\mu} & TTI \\
 \downarrow TTq_1 & \searrow \mu & \downarrow \mu & \searrow \mu & \downarrow \mu \\
 & TS & \xrightarrow{Tq_2} & TTI & \\
 & \downarrow Tq_1 & & & \\
 TTTI & \xrightarrow{\mu} & TTI & & TI \\
 T\mu \downarrow & \searrow \mu & & & \downarrow L \\
 TTI & \xrightarrow{\mu} & TI & \xrightarrow{L} & O
 \end{array}
 \qquad
 \begin{array}{l}
 \textcircled{1} \text{ naturality of } \mu \\
 \textcircled{2} \text{ monad law}
 \end{array}$$

As for reflexivity, $(\mu_I \circ Tq_1, \mu_I \circ Tq_2)$ is the composition of the reflexive pairs (μ_I, μ_I) and (Tq_1, Tq_2) . ◀

► **Lemma 5.4.** *Suppose T maps reflexive coequalisers to epimorphisms. If $i: B \rightarrow UC$ is such that $U(i^\sharp)$ reflexively coequalises $q_1, q_2: A \rightarrow TB$ in \mathcal{C} , then i^\sharp reflexively coequalises $q_1^\sharp, q_2^\sharp: FA \rightarrow FB$.*

Proof. For $k \in \{1, 2\}$ we have $U(i^\sharp \circ q_k^\sharp) \circ \eta_A = \text{rch} \circ U(q_k^\sharp) \circ \eta_A = \text{rch} \circ q_k$, so by $U(i^\sharp)$ coequalising q_1 and q_2 we have $i^\sharp \circ q_1^\sharp = i^\sharp \circ q_2^\sharp$. If a T -algebra homomorphism $f: TB \rightarrow Z$ is such that $f \circ q_1^\sharp = f \circ q_2^\sharp$, then

$$Uf \circ q_1 = Uf \circ U(q_1^\sharp) \circ \eta_A = Uf \circ U(q_2^\sharp) \circ \eta_A = Uf \circ q_2,$$

which because $U(i^\sharp)$ coequalises q_1 and q_2 yields a unique function $u: UC \rightarrow UZ$ such that $u \circ \text{rch} = f$. Remains to show that u is a T -algebra homomorphism. Note that since $U(i^\sharp)$ is a reflexive coequaliser, $TU(i^\sharp)$ is an epi by assumption on T . Precomposing u with $U(i^\sharp)$ yields the T -algebra homomorphism f , so by $TU(i^\sharp)$ being an epi and Lemma A.1 we conclude u is a T -algebra homomorphism $C \rightarrow Z$. Reflexivity of the pair follows from Lemma 5.11. ◀

► **Proposition 5.7.** For $\mathbf{C} = \mathbf{Set}$ and T any finitary monad, every language $L: TI \rightarrow O$ has a Nerode equivalence given by

$$R = \{(u, v) \in TI \times TI \mid L \circ \mu \circ T[\text{id}_{TI}, 1_u] = L \circ \mu \circ T[\text{id}_{TI}, 1_v]: T(TI + 1) \rightarrow O\}$$

with the corresponding projections $p_1, p_2: R \rightarrow TI$.

Proof. For each subset $X \subseteq R$, we define $p_X: R \rightarrow TI$ by

$$p_X(r) = \begin{cases} p_1(r) & \text{if } r \notin X \\ p_2(r) & \text{if } r \in X. \end{cases}$$

We have $Tp_1 = Tp_0$ by definition. Consider any $t \in TR$ and let a finite $E \subseteq R$ with inclusion map $e: E \rightarrow R$ and $t' \in TE$ be such that $T(e)(t') = t$. These exist because T is finitary. We will show by induction on E that

$$(L \circ \mu \circ Tp_0)(t) = (L \circ \mu \circ Tp_E)(t). \quad (3)$$

The case where $E = \emptyset$ is clear, so assume $E = E' \cup \{z\}$ with $z \notin E'$ and (3) holds when E' is substituted for E . We fix the singleton $1 = \{\square\}$ and define $d: R \rightarrow TI + 1$ by

$$d(r) = \begin{cases} (\kappa_1 \circ p_1)(r) & \text{if } r \notin E \\ (\kappa_1 \circ p_2)(r) & \text{if } r \in E' \\ \kappa_2(\square) & \text{if } r = z, \end{cases}$$

where κ_1 and κ_2 are the coproduct injections. By this definition, we have $[\text{id}_{TI}, 1_{p_1(z)}] \circ d = p_{E'}$ and $[\text{id}_{TI}, 1_{p_2(z)}] \circ d = p_E$, so

$$\begin{aligned} (L \circ \mu \circ Tp_0)(t) &= (L \circ \mu \circ Tp_{E'})(t) && \text{(induction hypothesis)} \\ &= (L \circ \mu \circ T([\text{id}_{TI}, 1_{p_1(z)}] \circ d))(t) \\ &= (L \circ \mu \circ T([\text{id}_{TI}, 1_{p_2(z)}] \circ d))(t) && \text{(definition of } R) \\ &= (L \circ \mu \circ Tp_E)(t), \end{aligned}$$

thus concluding the proof of (3). Now $Tp_1 = Tp_0$ by definition and

$$Tp_E(t) = T(p_E \circ e)(t') = T(p_2 \circ e)(t') = Tp_2(t),$$

from which we find that $(L \circ \mu \circ Tp_1)(t) = (L \circ \mu \circ Tp_0)(t) = (L \circ \mu \circ Tp_E)(t) = (L \circ \mu \circ Tp_2)(t)$. As this argument works for any $t \in TR$, we have $L \circ \mu \circ Tp_1 = L \circ \mu \circ Tp_2$.

Now consider any set S with $q_1, q_2: S \rightarrow TI$ making

$$\begin{array}{ccc} TS & \xrightarrow{Tq_2} & TTI \\ \downarrow Tq_1 & & \downarrow \mu \\ & & TI \\ & & \downarrow L \\ TTI & \xrightarrow{\mu} & TI \xrightarrow{L} O \end{array} \quad (4)$$

commute, and assume q_1 and q_2 have a common section $j: TI \rightarrow S$. We define $u: S \rightarrow R$ by $u(s) = (q_1(s), q_2(s))$. To see that this is indeed an element of R , note that for $k \in \{1, 2\}$,

$$\begin{aligned} L \circ \mu \circ T[\text{id}_{TI}, 1_{q_k(s)}] &= L \circ \mu \circ T[\text{id}_{TI}, q_k \circ 1_s] \\ &= L \circ \mu \circ T[q_k \circ j, q_k \circ 1_s] && \text{(section)} \\ &= L \circ \mu \circ Tq_k \circ T[j, 1_s], \end{aligned}$$

and therefore $L \circ \mu \circ T[\text{id}_{TI}, 1_{q_1(s)}] = L \circ \mu \circ T[\text{id}_{TI}, 1_{q_2(s)}]$ follows from (4). By definition, u is the unique map making the diagram below commute.

$$\begin{array}{ccc} & S & \\ q_1 \swarrow & \downarrow u & \searrow q_2 \\ TI \xleftarrow{p_1} & R & \xrightarrow{p_2} TI \end{array}$$

► **Lemma 5.8.** *If \mathcal{C} has coproducts, then for any Nerode equivalence (R, p_1, p_2) there exists a unique T -algebra structure $u: TR \rightarrow R$ making p_1 and p_2 T -algebra homomorphisms $(R, u) \rightarrow (TI, \mu)$ that have a common section.*

Proof. Let $L: TI \rightarrow O$ be a language with Nerode equivalence (R, p_1, p_2) . Then (p_1, p_2) is a reflexive pair by the Nerode equivalence property, since $(\text{id}_{TI}, \text{id}_{TI})$ is a reflexive pair trivially satisfying the Nerode equivalence condition. We apply Lemma A.2 to obtain from the Nerode equivalence property a unique morphism $r: TR \rightarrow R$ making the diagram below commute.

$$\begin{array}{ccccc} TTI & \xleftarrow{Tp_1} & TR & \xrightarrow{Tp_2} & TTI \\ \downarrow \mu & & \downarrow r & & \downarrow \mu \\ TTI & \xleftarrow{p_1} & R & \xrightarrow{p_2} & TI \end{array} \quad (5)$$

We need to show that (R, r) is a T -algebra. The first commutative diagram below shows that $r \circ \eta_R$ preserves p_1 and p_2 , so since id_R also does this we must have $r \circ \eta_R = \text{id}_R$ by the uniqueness property of the Nerode equivalence.

$$\begin{array}{ccccc} & R & & & \\ & \swarrow p_1 & & \searrow p_2 & \\ & TI & \textcircled{2} & TI & \\ & \downarrow \eta & & \downarrow \eta & \\ & TTI & \xleftarrow{Tp_1} & TR & \xrightarrow{Tp_2} & TTI & \textcircled{1} \\ & \downarrow \mu & & \downarrow r & & \downarrow \mu & \\ TI & \xleftarrow{p_1} & R & \xrightarrow{p_2} & TI \end{array}$$

① monad law
 ② naturality of η
 ③ naturality of μ

$$\begin{array}{ccccc} TTTI & \xleftarrow{TTp_1} & TTR & \xrightarrow{TTp_2} & TTTI & & TTTI & \xleftarrow{TTp_1} & TTR & \xrightarrow{TTp_2} & TTTI \\ T\mu \downarrow & (5) & \downarrow T_r & (5) & \downarrow T\mu & & T\mu \downarrow & \mu & \downarrow \mu & \mu & \downarrow T\mu \\ TTI & \xleftarrow{Tp_1} & TR & \xrightarrow{Tp_2} & TTI & & TTI & \textcircled{1} & TTI & \xleftarrow{Tp_1} & TR & \xrightarrow{Tp_2} & TTI & \textcircled{1} & TTI \\ \mu \downarrow & (5) & \downarrow r & (5) & \downarrow \mu & & \mu \downarrow & \mu & \downarrow r & (5) & \downarrow \mu & \mu & \downarrow \mu \\ TI & \xleftarrow{p_1} & R & \xrightarrow{p_2} & TI & & TI & \xleftarrow{p_1} & R & \xrightarrow{p_2} & TI \end{array}$$

As for the other two, we use a double application of Lemma A.2 to see that the pair $(\mu \circ T\mu \circ TTp_1, \mu \circ T\mu \circ TTp_2)$ satisfies the Nerode equivalence conditions. Commutativity of the two diagrams then shows that both $r \circ T_r$ and $r \circ \mu$ are the unique map commuting with the pairs $(\mu \circ T\mu \circ TTp_1, \mu \circ T\mu \circ TTp_2)$ and (p_1, p_2) , so they must be equal and (TR, r) is a T -algebra.

It remains to show that p_1 and p_2 have a common section in $\mathcal{EM}(T)$. To this end, note that $([\eta_I, \text{id}_{TI}], [\eta_I, \text{id}_{TI}])$ is a reflexive pair trivially satisfying the Nerode equivalence condition. Thus, we obtain by the Nerode equivalence property a unique morphism $u: I + TI \rightarrow R$ making

$$\begin{array}{ccc} & I + TI & \\ [\eta, \text{id}] \swarrow & \downarrow u & \searrow [\eta, \text{id}] \\ TI \xleftarrow{p_1} & R & \xrightarrow{p_2} TI \end{array}$$

6:22 Tree Automata as Algebras: Minimisation and Determinisation

commute. Then for $k \in \{1, 2\}$,

$$p_k \circ (u \circ \kappa_1)^\sharp = (p_k \circ u \circ \kappa_1)^\sharp = (p_k \circ [\eta_I, \text{id}_{TI}])^\sharp = \eta_I^\sharp = \text{id}_{(TI, \mu)}. \quad \blacktriangleleft$$

► **Lemma 5.11.** *If $q_1, q_2: A \rightarrow TB$ is a reflexive pair in \mathbf{C} , then so is (q_1^\sharp, q_2^\sharp) in $\mathcal{EM}(T)$.*

Proof. Assume $j: TB \rightarrow A$ is the common section of q_1 and q_2 . Then, for $k \in \{1, 2\}$,

$$q_k^\sharp \circ (\eta_A \circ j \circ \eta_B)^\sharp = U((U(q_k^\sharp) \circ \eta_A \circ j \circ \eta_B)^\sharp) = U((q_k \circ j \circ \eta_B)^\sharp) = U(\eta_B^\sharp) = \text{id}_{TB}. \quad \blacktriangleleft$$

Coalgebraic Geometric Logic

Nick Bezhanishvili

Institute of Logic, Language and Computation, University of Amsterdam, The Netherlands
n.bezhanishvili@uva.nl

Jim de Groot 

Department of Engineering and Computer Science, The Australian National University,
Canberra, Australia
jim.degroot@anu.edu.au

Yde Venema

Institute of Logic, Language and Computation, University of Amsterdam, The Netherlands
y.venema@uva.nl

Abstract

Using the theory of coalgebra, we introduce a uniform framework for adding modalities to the language of propositional geometric logic. Models for this logic are based on coalgebras for an endofunctor \mathbf{T} on some full subcategory of the category \mathbf{Top} of topological spaces and continuous functions. We compare the notions of modal equivalence, behavioural equivalence and bisimulation on the resulting class of models, and we provide a final object for the corresponding category. Furthermore, we specify a method of lifting an endofunctor on \mathbf{Set} , accompanied by a collection of predicate liftings, to an endofunctor on the category of topological spaces.

2012 ACM Subject Classification Theory of computation \rightarrow Modal and temporal logics

Keywords and phrases Coalgebra, Geometric Logic, Modal Logic, Topology

Digital Object Identifier 10.4230/LIPIcs.CALCO.2019.7

Related Version A report version of the paper is available at <https://arxiv.org/abs/1903.08837>.¹

Acknowledgements The authors want to express their gratitude to the anonymous referees for many constructive and helpful comments.

1 Introduction

Propositional geometric logic arose at the interface of (pointfree) topology, logic and theoretical computer science as the logic of *finite observations* [1, 28]. Its language is constructed from a set of proposition letters by applying finite conjunctions and arbitrary disjunctions, these being the propositional operations preserving the property of finite observability. Through an interesting topological connection, formulas of geometric logic can be interpreted in the frame of open sets of a topological space. Central to this connection is the well-known adjunction between the category \mathbf{Frm} of frames and frame morphisms and the category \mathbf{Top} of topological spaces and continuous maps, which restricts to several interesting Stone-type dualities [15].

Coalgebraic logic is a framework in which generalised versions of modal logics are developed parametric in the signature of the language and a functor $\mathbf{T} : \mathbf{C} \rightarrow \mathbf{C}$ on some base category \mathbf{C} . With classical propositional logic as base logic, two natural choices for the base category are \mathbf{Set} , the category of sets and functions, and \mathbf{Stone} , the category of Stone spaces and continuous functions, i.e. the topological dual to the algebraic category of Boolean algebras.

¹ The presented material originates from the master's thesis of the second author, supervised by the first and third author [11].



Coalgebraic logic for endofunctors on **Set** has been well investigated and still is an active area of research, see e.g. [8, 20]. In this setting, modal operators can be defined using the notion of relation lifting [22] or predicate lifting [23]. Coalgebraic logic in the category of Stone coalgebras has been studied in [19, 13, 9], and there is a fairly extensive literature on the design of a coalgebraic modal logic based on a general Stone-type duality (or adjunction), see for instance [7] and references therein.

In this paper we investigate some links between coalgebraic logic and geometric logic. That is, we shall use methods from coalgebraic logic to introduce modal operators to the language of geometric logic, with the intention of studying interpretations of these logics in certain topological coalgebras. Note that extensions of geometric logic with the basic modalities \Box and \Diamond , which are closely related to the topological Vietoris construction, have received much attention in the literature, see [28] for some early history. A first step towards developing coalgebraic geometric logic was taken in [27], where a method is explored to lift a functor on **Set** to a functor on the category **KHaus** of compact Hausdorff spaces, and the connection is investigated between the lifted functor and a relation-lifting based “cover” modality.

Our aim here is to develop a framework for the coalgebraic geometric logics that arise if we extend geometric logic with modalities that are induced by appropriate *predicate liftings*. Guided by the connection between geometric logic and topological spaces, we choose the base category of our framework to be **Top** itself, or one of its full subcategories such as **Sob** (sober spaces), **KSob** (compact sober spaces) or **KHaus** (compact Hausdorff spaces). On this base category **C** we then consider an arbitrary endofunctor **T** which serves as the type of our topological coalgebras. Furthermore, we shall see that if we want our formulas to be interpreted as *open sets* of the coalgebra carrier, we need the predicate liftings that interpret the modalities of the language to satisfy some natural *openness* condition. Summarizing, we shall study the coalgebraic geometric logic induced by (1) a functor $\mathbf{T} : \mathbf{C} \rightarrow \mathbf{C}$, where **C** is a full subcategory of **Top**, and (2) a set Λ of open predicate liftings for **T**. As running examples we take the combination of the basic modalities for the Vietoris functor, and that of the monotone box and diamond modalities for various topological manifestations of the monotone neighborhood functor on **Set**. The structures providing the semantics for our coalgebraic geometric logics are the *T-models* consisting of a **T**-coalgebra together with a valuation mapping proposition letters to open sets in the coalgebra carrier.

The main results that we report on here are the following:

- In Section 4, we construct a final object in the category of **T**-models, where **T** is an endofunctor on **Top** which preserves sobriety and admits a Scott-continuous, characteristic geometric modal signature.
- After that, in Section 5 we adapt the method of [17], in order to lift a **Set**-functor together with a collection of predicate liftings to an endofunctor on **Top**. We obtain the Vietoris functor and monotone functor on **KHaus** as restrictions of such lifted functors.
- Finally, in Section 6 we transfer the notion of Λ -bisimilarity from [10, 2] to our setting, and we compare this to geometric modal equivalence, behavioural equivalence and Aczel-Mendler bisimilarity. Our main finding is that on the categories **Top**, **Sob** and **KSob**, the first three notions coincide, provided Λ and **T** meet some reasonable conditions.

We finish the paper with listing some questions for further research.

2 Preliminaries

We briefly fix notation and review some preliminaries.

Categories and functors

We use a **bold font** for categories. We assume familiarity with the following categories:

- **Set** is the category of sets and functions;
- **Top** is the category of topological spaces and continuous functions;
- **KHaus** and **Stone** are the full subcategories of **Top** whose objects are compact Hausdorff spaces and Stone spaces respectively;
- **BA** is the category of Boolean algebras and Boolean algebra homomorphisms.

Categories can be connected by functors. We use a **sans serif font** for functors. In particular, the following functors are regularly used in this paper:

- $U : \mathbf{Top} \rightarrow \mathbf{Set}$ is the forgetful functor sending a topological space to its underlying set. The functor U restricts to every subcategory of **Top**, in which case we shall abuse notation and also call it U ;
- $P : \mathbf{Set} \rightarrow \mathbf{Set}$ and $\check{P} : \mathbf{Set}^{\text{op}} \rightarrow \mathbf{Set}$ are the covariant and contravariant powerset functor respectively;
- $Q : \mathbf{Set}^{\text{op}} \rightarrow \mathbf{BA}$ sends a set to its Boolean powerset algebra and a function to the inverse image map viewed as morphism in **BA**;
- $\Omega : \mathbf{Top} \rightarrow \mathbf{Set}$ sends a topological space to the set of opens.

More categories and functors will be defined along the way. We use the symbol \equiv for categorical equivalence.

Coalgebra

Let \mathbf{C} be a category and T an endofunctor on \mathbf{C} . A T -coalgebra is a pair (X, γ) where X is an object in \mathbf{C} and $\gamma : X \rightarrow TX$ is a morphism in \mathbf{C} . A T -coalgebra morphism between two T -coalgebras (X, γ) and (X', γ') is a morphism $f : X \rightarrow X'$ in \mathbf{C} satisfying $\gamma' \circ f = Tf \circ \gamma$. The collection of T -coalgebras and T -coalgebra morphisms forms a category, which we shall denote by $\mathbf{Coalg}(T)$. The category \mathbf{C} is called the *base category* of $\mathbf{Coalg}(T)$.

► **Example 1** (Kripke frames). The category of Kripke frames and bounded morphisms is isomorphic to $\mathbf{Coalg}(P)$ [20].

► **Example 2** (Monotone neighbourhood frames). Let $D : \mathbf{Set} \rightarrow \mathbf{Set}$ be the functor given on objects by $DX = \{W \subseteq PX \mid \text{if } a \in W \text{ and } a \subseteq b \text{ then } b \in W\}$, for X a set. For a morphism $f : X \rightarrow X'$ define $Df : DX \rightarrow DX' : W \mapsto \{a' \in PX' \mid f^{-1}(a') \in W\}$. Then the category of monotone frames and bounded morphisms is isomorphic to $\mathbf{Coalg}(D)$ [6, 12, 13].

Coalgebraic logic for Set-coalgebras

Let T be a **Set**-functor and Φ a set of proposition letters. A T -model is a triple (X, γ, V) where (X, γ) is a T -coalgebra and $V : \Phi \rightarrow PX$ is a valuation of the proposition letters. An n -ary predicate lifting for T is a natural transformation $\lambda : \check{P}^n \rightarrow \check{P} \circ T$, where \check{P}^n denotes the n -fold product of the contravariant powerset functor. A predicate lifting is called *monotone* if for all sets X and subsets $a_1, \dots, a_n, b \subseteq X$ we have $\lambda_X(a_1, \dots, a_i, \dots, a_n) \subseteq \lambda_X(a_1, \dots, a_i \cup b, \dots, a_n)$. For a set Λ of predicate liftings for T , define the *language* $\text{ML}(\Lambda)$ by

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \heartsuit^\lambda(\varphi_1, \dots, \varphi_n),$$

7:4 Coalgebraic Geometric Logic

where $p \in \Phi$ and $\lambda \in \Lambda$ is n -ary. The *semantics* of $\varphi \in \text{ML}(\Lambda)$ on a T-model $\mathfrak{X} = (X, \gamma, V)$ is given recursively by $\llbracket p \rrbracket^{\mathfrak{X}} = V(p)$, $\llbracket \varphi_1 \wedge \varphi_2 \rrbracket^{\mathfrak{X}} = \llbracket \varphi_1 \rrbracket^{\mathfrak{X}} \cap \llbracket \varphi_2 \rrbracket^{\mathfrak{X}}$, $\llbracket \neg \varphi \rrbracket^{\mathfrak{X}} = X \setminus \llbracket \varphi \rrbracket^{\mathfrak{X}}$, and

$$\llbracket \lambda(\varphi_1, \dots, \varphi_n) \rrbracket^{\mathfrak{X}} = \gamma^{-1}(\lambda(\llbracket \varphi_1 \rrbracket^{\mathfrak{X}}, \dots, \llbracket \varphi_n \rrbracket^{\mathfrak{X}})),$$

where $p \in \Phi$ and λ ranges over Λ .

► **Example 3** (Kripke models). Consider for P-models the predicate liftings $\lambda^{\square}, \lambda^{\diamond} : \check{\mathbb{P}} \rightarrow \check{\mathbb{P}} \circ \mathbb{P}$ given by $\lambda_X^{\square}(a) = \{b \in \mathbb{P}X \mid b \subseteq a\}$ and $\lambda_X^{\diamond}(a) = \{b \in \mathbb{P}X \mid b \cap a \neq \emptyset\}$. Then λ^{\square} and λ^{\diamond} yield the usual Kripke semantics of \square and \diamond .

► **Example 4** (Monotone neighbourhood frames). Monotone neighbourhood models are precisely D-models, where D is the functor defined in Example 2. The usual semantics for the box and diamond in this setting can be obtained from the predicate liftings given by

$$\lambda_X^{\square}(a) = \{W \in \text{DX} \mid a \in W\}, \quad \lambda_X^{\diamond}(a) = \{W \in \text{DX} \mid X \setminus a \notin W\}. \quad (1)$$

We refer to [20] for many more examples of coalgebraic logic for **Set**-functors.

Frames and spaces

A *frame* is a complete lattice F in which for all $a \in F$ and $S \subseteq F$ we have $a \wedge \bigvee S = \bigvee \{a \wedge s \mid s \in S\}$. A *frame homomorphism* is a function between frames that preserves finite meets and arbitrary joins. For $a, b \in F$ we say that a is *well inside* b , notation: $a \triangleleft b$, if there is a $c \in F$ such that $c \wedge a = \perp$ and $c \vee b = \top$. An element $a \in F$ is called *regular* if $a = \bigvee \{b \in F \mid b \triangleleft a\}$ and a frame is called *regular* if all of its elements are regular. The *negation* of $a \in F$ is defined as $\sim a = \bigvee \{b \in F \mid a \wedge b = \perp\}$ and we have $a \triangleleft b$ iff $\sim a \vee b = \top$. A frame is said to be *compact* if $\bigvee S = \top$ implies that there is a finite subset $S' \subseteq S$ such that $\bigvee S' = \top$. Frames can be presented by generators and relations, and any presentation by generators and relations presents a unique frame. For details see [15, 28].

► **Remark 5.** We will regularly define a frame homomorphism $f : F \rightarrow F'$ from a frame F presented by $\langle G, R \rangle$ to some frame F' . It then suffices to give an assignment $f' : G \rightarrow F'$ such that whenever $x = x'$ is a relation in R , $f(x) = f(x')$ in F .

The collection of open sets of a topological space \mathcal{X} forms a frame, denoted $\text{opn}\mathcal{X}$. A continuous map $f : \mathcal{X} \rightarrow \mathcal{X}'$ induces $\text{opn}f = f^{-1} : \text{opn}\mathcal{X}' \rightarrow \text{opn}\mathcal{X}$ and with this definition opn is a contravariant functor $\mathbf{Top} \rightarrow \mathbf{Frm}$. A frame is called *spatial* if it is isomorphic to $\text{opn}\mathcal{X}$ for some topological space \mathcal{X} .

A *point* of a frame F is a frame homomorphism $p : F \rightarrow 2$, where $2 = \{\top, \perp\}$ is the two-element frame. Let $\text{pt}F$ be the collection of points of F endowed with the topology $\{\tilde{a} \mid a \in F\}$, where $\tilde{a} = \{p \in \text{pt}F \mid p(a) = \top\}$. For a frame homomorphism $f : F \rightarrow F'$ define $\text{pt}f : \text{pt}F' \rightarrow \text{pt}F$ by $p \mapsto p \circ f$. The assignment pt defines a contravariant functor $\mathbf{Frm} \rightarrow \mathbf{Top}$. A topological space that arises as the space of points of a lattice is called *sober*. The *sobrification* of a topological space \mathcal{X} is $\text{pt}(\text{opn}\mathcal{X})$.

We denote by **Sob** and **KSob** the full subcategories of **Top** whose objects are sober spaces and compact sober spaces, respectively. Where **Frm** is the category of frames and frame homomorphisms, **SFrm**, **KSFrm** and **KRFrm** are the full subcategories of **Frm** whose objects are spatial frames, compact spatial frames and compact regular frames, respectively. The functor $Z : \mathbf{Frm} \rightarrow \mathbf{Set}$ is the forgetful functor sending a frame to the underlying set, and restricts to every subcategory of **Frm**. Note that $\Omega = Z \circ \text{opn}$.

► **Fact 6.** The functor pt is a right adjoint to opn . This adjunction restricts to the duality $\mathbf{SFrm} \equiv \mathbf{Sob}^{\text{op}}$, which in turn restricts to $\mathbf{KSFrm} \equiv \mathbf{KSob}^{\text{op}}$ and $\mathbf{KRFrm} \equiv \mathbf{KHaus}^{\text{op}}$.

For a more thorough exposition of frames and spaces, and a proof of the statements in Fact 6 we refer to section C1.2 of [16]. We explicitly mention one isomorphism which is part of this duality, for we will encounter it later on.

► **Remark 7.** Let \mathcal{X} be a sober space. Then Fact 6 entails that there is an isomorphism $\mathcal{X} \rightarrow \text{pt}(\text{opn}\mathcal{X})$. This isomorphism sends x to p_x , where $p_x : \text{opn}\mathcal{X} \rightarrow 2$ is the point given by $p_x(a) = \top$ iff $x \in a$, for all $x \in \mathcal{X}$ and $a \in \Omega\mathcal{X}$.

3 Logic for topological coalgebras

Although not all of our results can be proved for every full subcategory of **Top**, we will give the basic definitions in full generality. To this end, we let \mathbf{C} be some full subcategory of **Top** and define coalgebraic logic over base category \mathbf{C} . In particular $\mathbf{C} = \mathbf{KHaus}$ and $\mathbf{C} = \mathbf{Sob}$ will be of interest. Throughout this section \mathbb{T} is an arbitrary endofunctor on \mathbf{C} . Recall that Φ is an arbitrary but fixed set of proposition letters. We commence with defining the topological version of a predicate lifting, called an open predicate lifting.

► **Definition 8.** An *open predicate lifting* for \mathbb{T} is a natural transformation

$$\lambda : \Omega^n \rightarrow \Omega \circ \mathbb{T}.$$

An open predicate lifting is called *monotone in its i -th argument* if for every $\mathcal{X} \in \mathbf{C}$ and all $a_1, \dots, a_n, b \in \Omega\mathcal{X}$ we have $\lambda_{\mathcal{X}}(a_1, \dots, a_i, \dots, a_n) \subseteq \lambda_{\mathcal{X}}(a_1, \dots, a_i \cup b, \dots, a_n)$, and *monotone* if it is monotone in every argument. It is called *Scott-continuous* in its i -th argument if for every \mathcal{X} and every directed set $A \subseteq \Omega\mathcal{X}$ we have $\lambda_{\mathcal{X}}(a_1, \dots, \bigcup A, \dots, a_n) = \bigcup_{b \in A} \lambda_{\mathcal{X}}(a_1, \dots, b, \dots, a_n)$ and *Scott-continuous* if it is Scott-continuous in every argument.

A collection of open predicate liftings for \mathbb{T} is called a *geometric modal signature* for \mathbb{T} . A geometric modal signature for a functor \mathbb{T} is called *monotone* if every open predicate lifting in it is monotone, *Scott-continuous* if every predicate lifting in it is Scott-continuous, and *characteristic* if for every topological space \mathcal{X} in \mathbf{C} the collection $\{\lambda_{\mathcal{X}}(a_1, \dots, a_n) \mid \lambda \in \Lambda \text{ } n\text{-ary}, a_i \in \Omega\mathcal{X}\}$ is a sub-base for the topology on $\mathbb{T}\mathcal{X}$.

► **Remark 9.** Using the fact that for any two (open) sets a, b the set $\{a, a \cup b\}$ is directed, it is easy to see that Scott-continuity implies monotonicity.

Scott-continuity will play a rôle in Section 4, where it is used to show that the collection of formulas modulo (semantic) equivalence is a set, rather than a proper class.

Let \mathcal{S} be the Sierpinski space, i.e. the two-element set $2 = \{0, 1\}$ topologised by $\{\emptyset, \{1\}, 2\}$. For a topological space \mathcal{X} and $a \subseteq \mathcal{X}$ let $\chi_a : \mathcal{X} \rightarrow \mathcal{S}$ be the characteristic map (i.e. $\chi_a(x) = 1$ iff $x \in a$). Note that χ_a is continuous if and only if $a \in \Omega\mathcal{X}$. Analogously to predicate liftings for **Set**-functors [25, Proposition 43], one can classify n -ary predicate liftings as open subsets of $\mathbb{T}\mathcal{S}^n$. This elucidates the analogy with predicate liftings for **Set**-functors.

► **Proposition 10.** Suppose $\mathcal{S} \in \mathbf{C}$, then there is a bijective correspondence between n -ary open predicate liftings and elements of $\Omega\mathbb{T}\mathcal{S}^n$. This correspondence is given as follows: To an open predicate lifting λ assign the set $\lambda_{\mathcal{S}^n}(\pi_1^{-1}(\{1\}), \dots, \pi_n^{-1}(\{1\})) \in \Omega\mathbb{T}\mathcal{S}^n$, where $\pi_i : \mathcal{S}^n \rightarrow \mathcal{S}$ be the i -th projection, and conversely, for $c \in \Omega\mathbb{T}\mathcal{S}^n$ define $\lambda^c : \Omega^n \rightarrow \Omega\mathbb{T}$ by $\lambda^c_{\mathcal{X}}(a_1, \dots, a_n) = (\mathbb{T}\{\chi_{a_1}, \dots, \chi_{a_n}\})^{-1}(c)$.

► **Definition 11.** The *language* induced by a geometric modal signature Λ is the collection $\text{GML}(\Lambda)$ of formulas defined by the grammar

$$\varphi ::= \top \mid p \mid \varphi_1 \wedge \varphi_2 \mid \bigvee_{i \in I} \varphi_i \mid \heartsuit^\lambda(\varphi_1, \dots, \varphi_n),$$

7:6 Coalgebraic Geometric Logic

where p ranges over the set Φ of proposition letters, I is some index set and $\lambda \in \Lambda$ is n -ary. Abbreviate $\perp := \bigvee \emptyset$. We call a formula in $\text{GML}(\Lambda)$ *finitary* if it does not involve any infinite disjunctions.

The language $\text{GML}(\Lambda)$ is interpreted in so-called geometric \mathbb{T} -models.

► **Definition 12.** A *geometric \mathbb{T} -model* is a triple $\mathfrak{X} = (\mathcal{X}, \gamma, V)$ where (\mathcal{X}, γ) is a \mathbb{T} -coalgebra and $V : \Phi \rightarrow \Omega\mathcal{X}$ is a valuation of the proposition letters. A map $f : \mathcal{X} \rightarrow \mathcal{X}'$ is a *geometric \mathbb{T} -model morphism* from (\mathcal{X}, γ, V) to $(\mathcal{X}', \gamma', V')$ if f is a coalgebra morphism between the underlying coalgebras and $f^{-1} \circ V' = V$. The collection of geometric \mathbb{T} -models and geometric \mathbb{T} -model morphisms forms a category, which we denote by $\mathbf{Mod}(\mathbb{T})$.

The *semantics* of $\varphi \in \text{GML}(\Lambda)$ on such a model $\mathfrak{X} = (\mathcal{X}, \gamma, V)$ is given recursively by

$$\begin{aligned} \llbracket \top \rrbracket^{\mathfrak{X}} &= X, & \llbracket p \rrbracket^{\mathfrak{X}} &= V(p), & \llbracket \varphi \wedge \psi \rrbracket^{\mathfrak{X}} &= \llbracket \varphi \rrbracket^{\mathfrak{X}} \cap \llbracket \psi \rrbracket^{\mathfrak{X}}, & \llbracket \bigvee_{i \in I} \varphi_i \rrbracket^{\mathfrak{X}} &= \bigcup_{i \in I} \llbracket \varphi_i \rrbracket^{\mathfrak{X}}, \\ \llbracket \heartsuit^\lambda(\varphi_1, \dots, \varphi_n) \rrbracket^{\mathfrak{X}} &= \gamma^{-1}(\lambda_{\mathcal{X}}(\llbracket \varphi_1 \rrbracket^{\mathfrak{X}}, \dots, \llbracket \varphi_n \rrbracket^{\mathfrak{X}})). \end{aligned}$$

We write $\mathfrak{X}, x \Vdash \varphi$ iff $x \in \llbracket \varphi \rrbracket^{\mathfrak{X}}$. Two states x and x' are called *modally equivalent* if they satisfy the same formulas, notation: $x \equiv_{\Lambda} x'$.

The following proposition shows that morphisms preserve truth. Its proof is similar to the proof of theorem 6.17 in [26].

► **Proposition 13.** Let Λ be a geometric modal signature for \mathbb{T} . Let $\mathfrak{X} = (\mathcal{X}, \gamma, V)$ and $\mathfrak{X}' = (\mathcal{X}', \gamma', V')$ be geometric \mathbb{T} -models and let $f : \mathfrak{X} \rightarrow \mathfrak{X}'$ be a geometric \mathbb{T} -model morphism. Then for all $\varphi \in \text{GML}(\Lambda)$ and $x \in \mathcal{X}$ we have $\mathfrak{X}, x \Vdash \varphi$ iff $\mathfrak{X}', f(x) \Vdash \varphi$.

We state the notion of behavioural equivalence for future reference.

► **Definition 14.** Let $\mathfrak{X} = (\mathcal{X}, \gamma, V)$ and $\mathfrak{X}' = (\mathcal{X}', \gamma', V')$ be two geometric \mathbb{T} -models and $x \in \mathcal{X}$, $x' \in \mathcal{X}'$ two states. We say that x and x' are *behaviourally equivalent in $\mathbf{Mod}(\mathbb{T})$* ($x \simeq_{\mathbf{Mod}(\mathbb{T})} x'$) if there exists a cospan $\mathfrak{X} \xrightarrow{f} \mathfrak{Y} \xleftarrow{f'} \mathfrak{X}'$ in $\mathbf{Mod}(\mathbb{T})$ such that $f(x) = f'(x')$.

As an immediate consequence of Proposition 13 we find that behavioural equivalence implies modal equivalence. Let us give some concrete examples of functors.

► **Example 15 (Trivial functor).** Let $\mathbf{2} = \{0, 1\}$ be topologised by $\{\emptyset, \{0, 1\}\}$ (the trivial topology). Define the functor $F : \mathbf{Top} \rightarrow \mathbf{Top}$ by $F\mathcal{X} = \mathbf{2}$ for every $\mathcal{X} \in \mathbf{Top}$ and $Ff = \text{id}_{\mathbf{2}}$, the identity map on $\mathbf{2}$, for every continuous function f . This is clearly a functor. Consider the open predicate lifting $\lambda : \Omega \rightarrow \Omega \circ F$ given by $\lambda_{\mathcal{X}}(a) = \mathbf{U}\mathbf{2}$ for all $a \in \Omega\mathcal{X}$. For a F -model $\mathfrak{X} = (\mathcal{X}, \gamma, V)$ we then have $\mathfrak{X}, x \Vdash \heartsuit^\lambda \varphi$ iff $\gamma(x) \in \lambda(\llbracket \varphi \rrbracket^{\mathfrak{X}})$ iff $\llbracket \varphi \rrbracket^{\mathfrak{X}} \in \Omega\mathcal{X}$. So $\heartsuit^\lambda = \top$.

Next we have a look at the Vietoris functor on \mathbf{KHaus} . Coalgebras for this functor have also been studied in [3].

► **Example 16 (Vietoris functor).** For a compact Hausdorff space \mathcal{X} , let $\mathbf{V}_{\mathbf{kh}}\mathcal{X}$ be the collection of closed subsets of \mathcal{X} topologised by the subbase

$$\boxplus a := \{b \in \mathbf{V}_{\mathbf{kh}}\mathcal{X} \mid b \subseteq a\}, \quad \boxtimes a := \{b \in \mathbf{V}_{\mathbf{kh}}\mathcal{X} \mid a \cap b \neq \emptyset\},$$

where a ranges over $\Omega\mathcal{X}$. For a continuous map $f : \mathcal{X} \rightarrow \mathcal{X}'$ define $\mathbf{V}_{\mathbf{kh}}f : \mathbf{V}_{\mathbf{kh}}\mathcal{X} \rightarrow \mathbf{V}_{\mathbf{kh}}\mathcal{X}'$ by $\mathbf{V}_{\mathbf{kh}}f(a) = f[a]$. If \mathcal{X} is compact Hausdorff, then so is $\mathbf{V}_{\mathbf{kh}}\mathcal{X}$ [21, Theorem 4.9], and if $f : \mathcal{X} \rightarrow \mathcal{X}'$ is a continuous map between compact Hausdorff spaces, then $\mathbf{V}_{\mathbf{kh}}f$ is well defined and continuous [19, Lemma 3.8], so $\mathbf{V}_{\mathbf{kh}}$ defines an endofunctor on \mathbf{KHaus} .

Let $\mathfrak{X} = (\mathcal{X}, \gamma, V)$ be a $\mathbf{V}_{\mathbf{kh}}$ -model. The natural transformation λ^\square defined by

$$\lambda_{\mathcal{X}}^\square : \Omega\mathcal{X} \rightarrow \Omega(\mathbf{V}_{\mathbf{kh}}\mathcal{X}) : a \mapsto \{b \in \mathbf{V}_{\mathbf{kh}}\mathcal{X} \mid b \subseteq a\},$$

where $\mathcal{X} \in \mathbf{Top}$, is such that $\mathfrak{X}, x \Vdash \heartsuit^{\lambda^\square} \varphi$ iff $\mathfrak{X}, x \Vdash \square \varphi$ (with the usual interpretation of \square). Similarly $\lambda_{\mathcal{X}}^\diamond : \Omega\mathcal{X} \rightarrow \Omega \circ \mathbf{V}_{\mathbf{kh}}\mathcal{X}$, given by $\lambda_{\mathcal{X}}^\diamond(a) = \diamond a$, yields the usual semantics of the diamond modality.

The functor defined in the next example generalises the monotone functor on **Stone** [13].

► **Example 17** (Monotone functor). For a compact Hausdorff space \mathcal{X} , let $\mathbf{D}_{\mathbf{kh}}\mathcal{X}$ be the collection of sets $W \subseteq \mathbf{P}X$ such that $u \in W$ iff there exists a closed $c \subseteq u$ such that every open superset of c is in W . Endow $\mathbf{D}_{\mathbf{kh}}\mathcal{X}$ with the topology generated by the subbase

$$\boxplus a := \{W \in \mathbf{D}_{\mathbf{kh}}\mathcal{X} \mid a \in W\}, \quad \diamond a := \{W \in \mathbf{D}_{\mathbf{kh}}\mathcal{X} \mid X \setminus a \notin W\},$$

where a ranges over $\Omega\mathcal{X}$. For continuous functions $f : \mathcal{X} \rightarrow \mathcal{X}'$ define $\mathbf{D}_{\mathbf{kh}}f : \mathbf{D}_{\mathbf{kh}}\mathcal{X} \rightarrow \mathbf{D}_{\mathbf{kh}}\mathcal{X}' : W \mapsto \{a \in \mathbf{P}X \mid f^{-1}(a) \in W\}$. It is proven in the report version of the current paper [4] that this defines an endofunctor on **KHaus** which naturally extends the monotone functor on **Stone** [13, 9]. The open predicate liftings $\lambda^\square, \lambda^\diamond : \Omega\mathcal{X} \rightarrow \Omega\mathbf{T}$ defined by $\lambda_{\mathcal{X}}^\square(a) = \boxplus a$ and $\lambda_{\mathcal{X}}^\diamond(a) = \diamond a$ yield the usual box and diamond semantics of monotone modal logic [12].

In Section 6 it turns out to be useful to have a slightly stronger notion of open predicate liftings, called strong open predicate liftings, as this allows us to prove that behavioural equivalence implies so-called Λ -bisimilarity. Whereas the action of open predicate liftings is defined only on open subsets, a strong open predicate lifting acts on *every* subset of elements of a topological space. Recall that $\mathbf{U} : \mathbf{Top} \rightarrow \mathbf{Set}$ is the forgetful functor.

► **Definition 18.** A *strong open predicate lifting* for $\mathbf{T} : \mathbf{C} \rightarrow \mathbf{C}$ is a natural transformation $\mu : (\check{\mathbf{P}} \circ \mathbf{U})^n \rightarrow \check{\mathbf{P}} \circ \mathbf{U} \circ \mathbf{T}$ such that for all $\mathcal{X} \in \mathbf{C}$ and $a_1, \dots, a_n \in \Omega\mathcal{X}$ the set $\lambda_{\mathcal{X}}(a_1, \dots, a_n)$ is open in $\mathbf{T}\mathcal{X}$. Monotonicity of strong open predicate liftings is defined in the standard way.

We call an open predicate lifting (from Definition 11) *strong* if it is the restriction of some strong open predicate lifting and *strongly monotone* if it is the restriction of a monotone strong open predicate lifting.

Evidently, every strong open predicate lifting restricts to an open predicate lifting, and it is only this weaker notion of open predicate lifting that has an effect on the semantics. Our notion of strong open predicate lifting is similar to the notion of a *topological predicate lifting* for endofunctors on **Stone**, which were introduced in [9].

► **Example 19.** The predicate lifting corresponding to the box modality from Example 16 is strong, for it is the restriction of $\mu : \mathbf{U} \rightarrow \mathbf{U} \circ \mathbf{V}_{\mathbf{kh}}$ given by $\mu_{\mathcal{X}}(u) = \{b \in \mathbf{V}_{\mathbf{kh}}\mathcal{X} \mid b \subseteq u\}$. Likewise, all other predicate liftings from Examples 15, 16 and 17 are strong as well.

We devote the remainder of this section to investigating strong open predicate liftings. Recall from Example 15 that $\mathbf{2}$ denotes the two-element set with the trivial topology. We claim that natural transformations $\mu : (\check{\mathbf{P}} \circ \mathbf{U})^n \rightarrow \check{\mathbf{P}} \circ \mathbf{U} \circ \mathbf{T}$ correspond one-to-one with elements of $\check{\mathbf{P}}\mathbf{UT}\mathbf{2}$, provided $\mathbf{2} \in \mathbf{C}$: To a natural transformation μ associate the set $\mu_{\mathbf{2}}(p_1^{-1}(\{1\}), \dots, p_n^{-1}(\{1\}))$, where $p_i : \mathbf{2}^n \rightarrow \mathbf{2}$ denotes the i -th projection. Conversely, for $c \in \check{\mathbf{P}}\mathbf{UT}\mathbf{2}$ define μ^c by $\mu_{\mathcal{X}}^c(a_1, \dots, a_n) = (\mathbf{T}\langle \chi'_{a_1}, \dots, \chi'_{a_n} \rangle)^{-1}(c)$, where \mathcal{X} is a topological space, $a \subseteq \mathbf{U}\mathcal{X}$ and $\chi'_a : \mathcal{X} \rightarrow \mathbf{2}$ is the characteristic map. Note that χ'_a is continuous regardless of whether a is open or not, hence \mathbf{T} acts on all χ'_a . Details of the bijection are left to the reader.

► **Proposition 20.** *Let \mathbb{T} be an endofunctor on \mathbf{C} and suppose that \mathbf{C} contains the spaces $\mathbf{2}$ and \mathcal{S} . Let $s : \mathcal{S} \rightarrow \mathbf{2}$ be the identity map and let $c \in \check{\mathbf{P}}\mathbf{UT}\mathbf{2}^n$. The natural transformation μ^c is a strong open predicate lifting if and only if $(\mathbb{T}s^n)^{-1}(c) \subseteq \mathbb{T}\mathcal{S}^n$ is open.*

Proof. We give the proof for the case $n = 1$, the general case being similar. Left to right follows from the fact that $\{1\}$ is open in \mathcal{S} , hence $\mu_{\mathcal{S}}^c(\{1\}) = (\mathbb{T}\chi'_{\{1\}})^{-1}(c) = (\mathbb{T}s)^{-1}(c)$ must be open in $\mathbb{T}\mathcal{S}$. For the converse, let \mathcal{X} be a topological space and $a \in \Omega\mathcal{X}$. We need to show that $\mu_{\mathcal{X}}^c(a)$ is open. Since a is open, the characteristic map $\chi_a : \mathcal{X} \rightarrow \mathcal{S}$ is continuous and hence $\chi'_a = s \circ \chi_a$. We have

$$\begin{aligned} \mu_{\mathcal{X}}^c(a) &= (\mathbb{T}\chi'_a)^{-1}(c) && \text{(definition of } \mu^c) \\ &= (\mathbb{T}(s \circ \chi_a))^{-1}(c) && (\chi'_a = s \circ \chi_a) \\ &= (\mathbb{T}s \circ \mathbb{T}\chi_a)^{-1}(c) && \text{(definition of functors)} \\ &= (\mathbb{T}\chi_a)^{-1} \circ (\mathbb{T}s)^{-1}(c). && \text{(definition of inverse)} \end{aligned}$$

Since $\mathbb{T}\chi_a$ is continuous and $(\mathbb{T}s)^{-1}(c)$ is assumed to be open in $\mathbb{T}\mathcal{S}$, the set $\mu_{\mathcal{X}}^c(a)$ is open in $\mathbb{T}\mathcal{X}$. ◀

The following proposition gives two sufficient conditions on \mathbb{T} for its open predicate liftings to be strong. For a full subcategory \mathbf{C} of **Top** let **preC** denote the category of topological spaces in \mathbf{C} and (not necessarily continuous) functions.

► **Proposition 21.** *Let \mathbb{T} be an endofunctor on \mathbf{C} and suppose $\mathbf{2}, \mathcal{S} \in \mathbf{C}$.*

1. *If \mathbb{T} preserves injective functions then every open predicate lifting for \mathbb{T} is strong.*
2. *If \mathbb{T} extends to **preC**, then every open predicate lifting for \mathbb{T} is strong.*

Proof. For the first item, let $c \in \Omega\mathbb{T}\mathcal{S}^n$ determine the n -ary open predicate lifting λ^c . Since s^n is injective, by assumption $\mathbb{T}s^n$ is as well, and hence $c = (\mathbb{U}\mathbb{T}s^n)^{-1}((\mathbb{U}\mathbb{T}s^n)[c])$. Proposition 20 now implies that $\mu^{(\mathbb{U}\mathbb{T}s^n)[c]}$ is a strong open predicate lifting. It is easy to see that $\mu^{(\mathbb{U}\mathbb{T}s^n)[c]}$ extends λ^c , hence the latter is strong.

For the second item we show that, under the assumption, \mathbb{T} preserves injective functions. Let $f : \mathcal{X} \rightarrow \mathcal{Y}$ be an injective function in \mathbf{C} , then there exists a (not necessarily continuous) function $g : \mathcal{Y} \rightarrow \mathcal{X}$ satisfying $g \circ f = \text{id}_{\mathcal{X}}$. Then $\mathbb{T}g \circ \mathbb{T}f = \mathbb{T}(g \circ f) = \mathbb{T}\text{id}_{\mathcal{X}} = \text{id}_{\mathbb{T}\mathcal{X}}$, so $\mathbb{T}f$ has a (set-theoretic) left-inverse, hence is injective. ◀

Monotone open predicate lifting for an endofunctor on **KHaus** are always strong:

► **Proposition 22.** *Let \mathbb{T} be an endofunctor on **KHaus** and Λ a monotone geometric modal signature for \mathbb{T} . Then Λ is strongly monotone.*

Proof. Let $\lambda \in \Lambda$. We need to show that λ is the restriction of some strong monotone predicate lifting. Define

$$\tilde{\lambda}_{\mathcal{X}} : \check{\mathbf{P}}^n\mathbf{U}\mathcal{X} \rightarrow \check{\mathbf{P}}\mathbf{UT}\mathcal{X} : (b_1, \dots, b_n) \mapsto \bigcap \{ \lambda_{\mathcal{X}}(a_1, \dots, a_n) \mid a_i \in \Omega\mathcal{X} \text{ and } a_i \supseteq b_i \}.$$

Monotonicity of $\lambda_{\mathcal{X}}$ ensures $\tilde{\lambda}_{\mathcal{X}}(a) = \lambda_{\mathcal{X}}(a)$ for all $a \in \Omega\mathcal{X}$ and $\tilde{\lambda}$ is monotone by construction. So we only need to show that $\tilde{\lambda}$ is indeed a strong open predicate lifting, i.e. a natural transformation $\check{\mathbf{P}}^n\mathbf{U}\mathcal{X} \rightarrow \check{\mathbf{P}}\mathbf{UT}\mathcal{X}$. We assume λ to be unary, the general case being similar.

For a continuous map $f : \mathcal{X} \rightarrow \mathcal{X}'$ between compact Hausdorff spaces we need to show that $\tilde{\lambda}_{\mathcal{X}} \circ f^{-1} = (\mathbb{T}f)^{-1} \circ \tilde{\lambda}_{\mathcal{X}'}$. Since, by naturality of λ , the right hand side is equal to $\bigcap \{ \lambda_{\mathcal{X}}(f^{-1}(a')) \mid a' \in \Omega\mathcal{X}' \text{ and } b' \subseteq a' \}$, it suffices to show

$$\bigcap \{ \lambda_{\mathcal{X}}(c) \mid c \in \Omega\mathcal{X} \text{ and } f^{-1}(b') \subseteq c \} = \bigcap \{ \lambda_{\mathcal{X}}(f^{-1}(a')) \mid a' \in \Omega\mathcal{X}' \text{ and } b' \subseteq a' \}. \quad (2)$$

If a' is an open superset of b' then clearly $f^{-1}(b') \subseteq f^{-1}(a')$. So every element in the intersection of the right hand side is contained in the one on the left hand side and therefore we have \subseteq in (2). For the converse, suppose $c \in \Omega\mathcal{X}$ and $f^{-1}(b') \subseteq c$. Then the set $a' = X' \setminus f[X \setminus c]$ is open, contains b' , and satisfies $f^{-1}(b') \subseteq f^{-1}(a') \subseteq c$. Therefore $\lambda_{\mathcal{X}}(f^{-1}(a'))$ is one of the elements in the intersection on the left hand side of (2). Since $\lambda_{\mathcal{X}}(f^{-1}(a')) \subseteq \lambda_{\mathcal{X}}(c)$ this shows “ \supseteq ” in (2). ◀

4 A final model

We construct a final model in $\mathbf{Mod}(\mathbb{T})$ for a functor \mathbb{T} where either \mathbb{T} is an endofunctor on \mathbf{Sob} , or \mathbb{T} is an endofunctor on \mathbf{Top} which preserves sobriety. This assumption need not be problematic: If a functor on \mathbf{Top} does not preserve sobriety we can look at its sobrification. Topological functors which arise as lifts from set functors using the procedure in Section 5 automatically preserve sobriety.

► **Assumption.** Throughout this section, fix an endofunctor \mathbb{T} on \mathbf{Top} which preserves sobriety, and a Scott-continuous characteristic geometric modal signature Λ for \mathbb{T} . Recall that Φ is a set of proposition letters.

► **Definition 23.** Call two formulas φ and ψ *equivalent* in $\mathbf{Mod}(\mathbb{T})$ with respect to Λ , notation: $\varphi \equiv_{\mathbb{T}, \Lambda} \psi$, if $\mathfrak{X}, x \Vdash \varphi$ iff $\mathfrak{X}, x \Vdash \psi$ for all $\mathfrak{X} \in \mathbf{Mod}(\mathbb{T})$ and $x \in \mathfrak{X}$. Denote the equivalence class of φ in $\mathbf{GML}(\Lambda)$ by $[\varphi]$. Let $\mathbf{E} = \mathbf{E}(\mathbb{T}, \Lambda, \Phi)$ be the collection of formulas modulo $\equiv_{\mathbb{T}, \Lambda}$.

Recall that a finitary formula is one which does not involve arbitrary disjunctions.

► **Lemma 24 (Normal form).** *Under the assumption, every formula is equivalent to a formula of the form $\bigvee_{i \in I} \varphi_i$, where all the φ_i are finitary formulas.*

Proof. The proof proceeds by induction on the complexity of the formula. Suppose $\varphi = \varphi_1 \vee \varphi_2$. By induction we may assume that $\varphi_1 \equiv_{\mathbb{T}, \Lambda} \bigvee_{i \in I} \psi_i$ and $\varphi_2 \equiv_{\mathbb{T}, \Lambda} \bigvee_{j \in J} \psi_j$, where all the ψ_i and ψ_j are finitary, and we have $\varphi \equiv_{\mathbb{T}, \Lambda} \bigvee_{i \in I \cup J} \psi_i$, as desired. If $\varphi = \varphi_1 \wedge \varphi_2$, then $\varphi \equiv_{\mathbb{T}, \Lambda} (\bigvee_{i \in I} \psi_i) \wedge (\bigvee_{j \in J} \psi_j) \equiv_{\mathbb{T}, \Lambda} \bigvee_{(i,j) \in I \times J} \psi_i \wedge \psi_j$. Lastly, suppose $\varphi = \heartsuit^\lambda (\bigvee_{i \in I} \psi_i)$, where all the ψ_i are finitary. Then we have $\bigvee_{i \in I} \psi_i = \bigvee \{ \bigvee_{i \in I'} \psi_i \mid I' \subseteq I \text{ finite} \}$ and by construction the set $\{ \llbracket \bigvee_{i \in I'} \psi_i \rrbracket^{\mathfrak{X}} \mid I' \subseteq I, I' \text{ finite} \}$ is directed for every \mathbb{T} -model $\mathfrak{X} = (\mathcal{X}, \gamma, V)$. Hence by Scott-continuity of λ we obtain

$$\lambda_{\mathcal{X}}(\llbracket \bigvee_{i \in I} \psi_i \rrbracket^{\mathfrak{X}}) = \lambda_{\mathcal{X}}(\bigcup \{ \llbracket \bigvee_{i \in I'} \psi_i \rrbracket^{\mathfrak{X}} \mid I' \subseteq I \text{ finite} \}) = \bigcup \{ \lambda_{\mathcal{X}}(\llbracket \bigvee_{i \in I'} \psi_i \rrbracket^{\mathfrak{X}} \mid I' \subseteq I \text{ finite} \}.$$

Therefore $\varphi \equiv_{\mathbb{T}, \Lambda} \bigvee \{ \heartsuit^\lambda (\bigvee_{i \in I} \psi_i) \mid I' \subseteq I \text{ finite} \}$, i.e. φ is equivalent to an arbitrary disjunction of finitary formulas. The case for n -ary modalities is similar. This proves the lemma. ◀

► **Corollary 25.** *The collection \mathbf{E} from Definition 23 is a set.*

Proof. This follows immediately from Lemma 24 and the fact that the collection of finitary formulas is a set. ◀

► **Definition 26.** Define disjunction and arbitrary conjunction on \mathbf{E} by $[\varphi] \wedge [\psi] := [\varphi \wedge \psi]$ and $\bigvee_{i \in I} [\varphi_i] := [\bigvee_{i \in I} \varphi_i]$. It is easy to check that \mathbf{E} is a frame.

Set $\mathbf{L} = \mathbf{opn} \circ \mathbb{T} \circ \mathbf{pt} : \mathbf{Frm} \rightarrow \mathbf{Frm}$. This functor restricts to an endofunctor on \mathbf{SFrm} which is dual to the restriction of \mathbb{T} to \mathbf{Sob} . Since Λ is characteristic, the frame \mathbf{LE} is generated

7:10 Coalgebraic Geometric Logic

by $\{\lambda_{\mathcal{X}}(\widetilde{[\varphi_1]}, \dots, \widetilde{[\varphi_n]}) \mid \lambda \in \Lambda, \varphi_i \in \text{GML}(\Lambda)\}$. Define an L-algebra structure $\delta : \mathbf{LE} \rightarrow \mathbf{E}$ on generators by

$$\delta : \mathbf{LE} \rightarrow \mathbf{E} : \lambda_{\text{ptE}}(\widetilde{[\varphi_1]}, \dots, \widetilde{[\varphi_n]}) \mapsto [\heartsuit^\lambda(\varphi_1, \dots, \varphi_n)].$$

To show that δ is well defined it suffices to verify that the images of the generators of \mathbf{E} satisfy the same relations that they satisfy in \mathbf{LE} . We refer to the report version of the current paper for details. The dual of \mathbf{E} will be the topological space underlying the final model in $\mathbf{Mod}(\mathbb{T})$:

► **Definition 27.** Set $\mathcal{Z} := \text{ptE}$ and let $\zeta : \mathcal{Z} \rightarrow \mathbb{T}\mathcal{Z}$ be the composition

$$\text{ptE} \xrightarrow{\text{pt}\delta} \text{pt}(\mathbf{LE}) \xlongequal{\quad} \text{pt}(\text{opn}(\mathbb{T}(\text{ptE}))) \xrightarrow{k_{\mathbb{T}(\text{ptE})}^{-1}} \mathbb{T}(\text{ptE}),$$

where $k_{\mathbb{T}(\text{ptE})} : \mathbb{T}(\text{ptE}) \rightarrow \text{pt}(\text{opn}(\mathbb{T}(\text{ptE})))$ is the isomorphism given in Remark 7. Together with the valuation $V_{\mathcal{Z}} : \Phi \rightarrow \Omega\mathcal{Z} : p \mapsto [p]$, the triple $\mathfrak{Z} = (\mathcal{Z}, \zeta, V_{\mathcal{Z}})$ forms a \mathbb{T} -model.

For an object $\Gamma \in \mathcal{Z}$, the element $(\text{pt}\delta)(\Gamma)$ is the completely prime filter

$$F = \{\lambda(\widetilde{\varphi_1}, \dots, \widetilde{\varphi_n}) \in \text{pt}(\text{opn}(\mathbb{T}(\text{ptE}))) \mid [\heartsuit^\lambda(\varphi_1, \dots, \varphi_n)] \in \Gamma\}$$

in $\text{pt}(\text{opn}(\mathbb{T}(\text{ptE})))$. The element $\zeta(\Gamma)$ is the unique element in $\mathbb{T}(\text{ptE})$ corresponding to F under the isomorphism $k_{\mathbb{T}(\text{ptE})}$. By definition of $k_{\mathbb{T}(\text{ptE})}$, this is the unique element in the intersection of $\{\lambda_{\text{ptE}}(\widetilde{[\varphi_1]}, \dots, \widetilde{[\varphi_n]}) \mid [\heartsuit^\lambda(\varphi_1, \dots, \varphi_n)] \in \Gamma\}$. Moreover, it follows from the definition of $k_{\mathbb{T}(\text{ptE})}$ that $[\heartsuit^\lambda(\varphi_1, \dots, \varphi_n)] \notin \Gamma$ implies $\zeta(\Gamma) \notin \lambda_{\text{ptE}}(\widetilde{[\varphi_1]}, \dots, \widetilde{[\varphi_n]})$. The following lemma follows from the previous discussion and a straightforward induction. Both Lemma 28 and Proposition 29 are proven in detail in the report version of this paper.

► **Lemma 28** (Truth lemma). *For all $\Gamma \in \mathcal{Z}$ we have $\mathfrak{Z}, \Gamma \Vdash \varphi$ iff $[\varphi] \in \Gamma$.*

► **Proposition 29.** *For every geometric \mathbb{T} -model $\mathfrak{X} = (\mathcal{X}, \gamma, V)$ the map $\text{th}_{\mathfrak{X}} : \mathcal{X} \rightarrow \mathcal{Z}$ given by $x \mapsto \{[\varphi] \in \mathbf{E} \mid \mathfrak{X}, x \Vdash \varphi\}$ is a \mathbb{T} -model morphism.*

The developed theory results in the following theorem.

► **Theorem 30.** *Let \mathbb{T} be a sobriety-preserving endofunctor on \mathbf{Top} and Λ a Scott-continuous characteristic geometric modal signature for \mathbb{T} . Then $\mathfrak{Z} = (\mathcal{Z}, \zeta, V_{\mathcal{Z}})$ is final in $\mathbf{Mod}(\mathbb{T})$.*

Proof. Proposition 29 states that for every geometric \mathbb{T} -model $\mathfrak{X} = (\mathcal{X}, \gamma, V)$ there exists a \mathbb{T} -coalgebra morphism $\text{th}_{\mathfrak{X}} : \mathfrak{X} \rightarrow \mathfrak{Z}$, so we only need to show that this morphism is unique. Let $f : \mathfrak{X} \rightarrow \mathfrak{Z}$ be any coalgebra morphism. Then by Proposition 13 and Lemma 28 we have $[\varphi] \in f(x)$ iff $\mathfrak{Z}, f(x) \Vdash \varphi$ iff $\mathfrak{X}, x \Vdash \varphi$ for all $x \in \mathcal{X}$, hence $f = \text{th}_{\mathfrak{X}}$. ◀

► **Theorem 31.** *Under the assumptions of Theorem 30, we have $\equiv_{\Lambda} = \simeq_{\mathbf{Mod}(\mathbb{T})}$.*

Proof. If x and x' are behaviourally equivalent, then they are modally equivalent by Proposition 13. Conversely, if they are modally equivalent, then $\text{th}_{\mathfrak{X}}(x) = \text{th}_{\mathfrak{X}'}(x')$ by construction, so they are behaviourally equivalent. ◀

► **Remark 32.** If \mathbb{T} is an endofunctor on \mathbf{Sob} rather than \mathbf{Top} , the same procedure yields a final model in $\mathbf{Mod}(\mathbb{T})$. In particular, \mathbb{T} need not be the restriction of a \mathbf{Top} -endofunctor. However, if \mathbb{T} is an endofunctor on \mathbf{KSob} or \mathbf{KHaus} the procedure above does not guarantee a final coalgebra in $\mathbf{Mod}(\mathbb{T})$; indeed the state space \mathcal{Z} of the final coalgebra \mathfrak{Z} we construct need not be compact sober or compact Hausdorff. Of course, there may be a different way to attain similar results for \mathbf{KSob} or \mathbf{KHaus} . We leave this as an interesting open question. In Theorem 51 we prove an analog of Theorem 31 for endofunctors on \mathbf{KSob} .

5 Lifting functors from Set to Top

In [18, Section 4] the authors give a method to lift a **Set**-functor $T : \mathbf{Set} \rightarrow \mathbf{Set}$, together with a collection of predicate liftings Λ for T , to an endofunctor on **Stone**. We adapt their approach to obtain an endofunctor \widehat{T}_Λ on **Top**. In this section the notation \bigvee^\uparrow is used for directed joins, i.e. joins over directed sets. To define the action of \widehat{T}_Λ on a topological space \mathcal{X} we take the following steps:

Step 1. Construct a frame $\dot{F}_\Lambda \mathcal{X}$ of the images of predicate liftings applied to the open sets of \mathcal{X} (viewed simply as subsets of $T(U\mathcal{X})$);

Step 2. Quotient $\dot{F}_\Lambda \mathcal{X}$ with a suitable relation that ensures $\bigvee_{b \in B}^\uparrow \lambda(b) = \lambda(\bigvee^\uparrow B)$ whenever λ is monotone;

Step 3. Employ the functor $\mathbf{pt} : \mathbf{Frm} \rightarrow \mathbf{Top}$ to obtain a (sober) topological space.

This is the content of Definitions 33, 35 and 37. Recall that $U : \mathbf{Top} \rightarrow \mathbf{Set}$ is the forgetful functor and that Q is the contravariant functor sending a set to its Boolean powerset algebra.

► **Definition 33.** Let $T : \mathbf{Set} \rightarrow \mathbf{Set}$ be a functor and Λ a collection of predicate liftings for T . We define a contravariant functor $\dot{F}_\Lambda : \mathbf{Top} \rightarrow \mathbf{Frm}$. For a topological space \mathcal{X} let $\dot{F}_\Lambda \mathcal{X}$ be the subframe of $Q(T(U\mathcal{X}))$ generated by the set

$$\{\lambda_{U\mathcal{X}}(a_1, \dots, a_n) \mid \lambda \in \Lambda \text{ } n\text{-ary}, a_1, \dots, a_n \in \Omega\mathcal{X}\}.$$

That is, we close this set under finite intersections and arbitrary unions in $Q(T(U\mathcal{X}))$. For a continuous map $f : \mathcal{X} \rightarrow \mathcal{X}'$ let $\dot{F}_\Lambda f : \dot{F}_\Lambda \mathcal{X}' \rightarrow \dot{F}_\Lambda \mathcal{X}$ be the restriction of $Q(T(Uf))$ to $\dot{F}_\Lambda \mathcal{X}'$.

► **Lemma 34.** *The assignment \dot{F}_Λ defines a contravariant functor.*

Proof. We need to show that \dot{F}_Λ is well defined on morphisms and that it is functorial. To show that the action of \dot{F}_Λ on morphisms is well-defined, it suffices to show that $(\dot{F}_\Lambda f)(\lambda_{U\mathcal{X}}(a'_1, \dots, a'_n)) \in \dot{F}_\Lambda(\mathcal{X})$ for all generators $\lambda_{U\mathcal{X}}(a'_1, \dots, a'_n)$ of $\dot{F}_\Lambda \mathcal{X}'$, because frame homomorphisms preserve finite meets and all joins. This holds by naturality of λ :

$$(\dot{F}_\Lambda f)(\lambda_{U\mathcal{X}}(a_1, \dots, a_n)) = (Tf)^{-1}(\lambda_{U\mathcal{X}'}(a_1, \dots, a_n)) = \lambda_{U\mathcal{X}}(f^{-1}(a_1), \dots, f^{-1}(a_n)).$$

By continuity of f we have $f^{-1}(a_i) \in \Omega\mathcal{X}$ so the latter is indeed in $\dot{F}_\Lambda \mathcal{X}$. Functoriality of \dot{F}_Λ follows from functoriality of $Q \circ T \circ U$. ◀

► **Definition 35.** Let Λ be a collection of predicate liftings for a set functor T . For $\mathcal{X} \in \mathbf{Top}$, let $\widehat{F}_\Lambda \mathcal{X}$ be the quotient of $\dot{F}_\Lambda \mathcal{X}$ with respect to the congruence \sim generated by

$$\bigvee_{b \in B}^\uparrow \lambda(a_1, \dots, a_{i-1}, b, a_{i+1}, \dots, a_n) \sim \lambda(a_1, \dots, a_{i-1}, \bigvee^\uparrow B, a_{i+1}, \dots, a_n)$$

for all $a_i \in \Omega\mathcal{X}$, $B \subseteq \Omega\mathcal{X}$ directed, and $\lambda \in \Lambda$ monotone in its i -th argument. Write $q_{\mathcal{X}} : \dot{F}_\Lambda \mathcal{X} \rightarrow \widehat{F}_\Lambda \mathcal{X}$ for the quotient map and $[x]$ for the equivalence class in $\widehat{F}_\Lambda \mathcal{X}$ of an element $x \in \dot{F}_\Lambda \mathcal{X}$. For a continuous function $f : \mathcal{X} \rightarrow \mathcal{X}'$ define $\widehat{F}_\Lambda f : \widehat{F}_\Lambda \mathcal{X}' \rightarrow \widehat{F}_\Lambda \mathcal{X} : [\lambda_{U\mathcal{X}}(a_1, \dots, a_n)] \mapsto [\dot{F}_\Lambda(\lambda_{U\mathcal{X}}(a_1, \dots, a_n))]$.

Quotienting by the congruence from Definition 35 ensures that the lifted versions of monotone predicate liftings are Scott-continuous, see Proposition 43 below.

► **Lemma 36.** *The assignment \widehat{F}_Λ defines a contravariant functor.*

7:12 Coalgebraic Geometric Logic

Proof. We need to prove functoriality of \widehat{F}_Λ and that $\widehat{F}_\Lambda f$ is well defined for every continuous map $f : \mathcal{X} \rightarrow \mathcal{X}'$. In order to show that \widehat{F}_Λ is well defined, it suffices to show that $\dot{F}_\Lambda f$ is invariant under the congruence \sim . If $f : \mathcal{X} \rightarrow \mathcal{X}'$ is a continuous, then

$$\begin{aligned} & \bigvee^{\uparrow}_{b' \in B} (\dot{F}_\Lambda f)(\lambda_{\mathcal{U}\mathcal{X}'}(a'_1, \dots, a'_{i-1}, b', a'_{i+1}, \dots, a'_n)) \\ &= \bigvee^{\uparrow}_{b' \in B} (\top f)^{-1}(\lambda_{\mathcal{U}\mathcal{X}'}(a'_1, \dots, a'_{i-1}, b', a'_{i+1}, \dots, a'_n)) \\ &= \bigvee^{\uparrow}_{b' \in B} \lambda_{\mathcal{U}\mathcal{X}}(f^{-1}(a'_1), \dots, f^{-1}(a'_{i-1}), f^{-1}(b'), f^{-1}(a'_{i+1}), \dots, f^{-1}(a'_n)) \\ &\sim \lambda_{\mathcal{U}\mathcal{X}}(f^{-1}(a'_1), \dots, f^{-1}(a'_{i-1}), f^{-1}(\bigvee^{\uparrow} B), f^{-1}(a'_{i+1}), \dots, f^{-1}(a'_n)) \\ &= \dot{F}_\Lambda f(\lambda_{\mathcal{U}\mathcal{X}}(a'_1, \dots, a'_{i-1}, \bigvee^{\uparrow} B, a'_{i+1}, \dots, a'_n)) \end{aligned}$$

so $\dot{F}_\Lambda f$ is invariant under the congruence. In the \sim -step we use the fact that $\{f^{-1}(b') \mid b' \in B\}$ is directed in $\Omega\mathcal{X}$. Functoriality of $\widehat{F}_\Lambda f$ follows from functoriality of $\mathbb{Q} \circ \top \circ \mathbb{U}$. \blacktriangleleft

We are now ready to define the topological Kupke-Kurz-Pattinson lift of a functor on **Set** together with a collection of predicate liftings, to a functor on **Top**.

► **Definition 37.** Define the *topological Kupke-Kurz-Pattinson lift* (KKP lift for short) of \top with respect to Λ to be the functor

$$\widehat{\top}_\Lambda = \text{pt} \circ \widehat{F}_\Lambda.$$

This is a functor **Top** \rightarrow **Top** and since pt lands in **Sob** it restricts to an endofunctor on **Sob**.

Let us put our theory into action. For details see the report version of the current paper.

► **Example 38** (The monotone functor). Recall the monotone functor \mathbb{D} on **Set** and the corresponding set of predicate liftings $\Lambda = \{\lambda^\square, \lambda^\diamond\}$ from Examples 2 and 4. It can be seen that the topological KKP lift $\widehat{\mathbb{D}}_\Lambda$ of \mathbb{D} with respect to Λ restricts to \mathbb{D}_{kh} .

► **Example 39** (The Vietoris functor). Likewise, one can show that, when restricted to **KHaus**, the topological KKP lift of \mathbb{P} with respect to the usual box and diamond lifting coincides with the Vietoris functor from Example 16.

► **Example 40.** Not every endofunctor on **Top** can be obtained as the lift of a **Set**-functor with respect to a (cleverly) chosen set of predicate liftings in the sense of Definition 37. A trivial counterexample is the functor $\mathbb{F} : \mathbf{Top} \rightarrow \mathbf{Top}$ from Example 15. For every topological space \mathcal{X} we have $\mathbb{F}\mathcal{X} = \mathbf{2}$, which is not a T_0 space, hence not a sober space. Therefore \mathbb{F} does not preserve sobriety, while every lifted functor automatically preserves sobriety. Thus \mathbb{F} is not the lift of a **Set**-functor.

We describe how to lift a predicate lifting to an open predicate lifting. Recall that $\mathbb{Z} : \mathbf{Frm} \rightarrow \mathbf{Set}$ is the forgetful functor which sends a frame to its underlying set.

► **Definition 41.** Let Λ be a collection of predicate liftings for a functor $\top : \mathbf{Set} \rightarrow \mathbf{Set}$. A predicate lifting $\lambda : \check{\mathbb{P}}^n \rightarrow \check{\mathbb{P}} \circ \top$ in Λ induces an open predicate lifting $\widehat{\lambda} : \Omega^n \rightarrow \Omega \circ \widehat{\top}$ for $\widehat{\top}$ via

$$\Omega^n \mathcal{X} \xrightarrow{\lambda_{\mathcal{U}\mathcal{X}}} \mathbb{Z}(\dot{F}_\Lambda \mathcal{X}) \xrightarrow{\mathbb{Z}q_{\mathcal{X}}} \mathbb{Z}(\widehat{F}_\Lambda \mathcal{X}) \xrightarrow{\mathbb{Z}k_{\widehat{F}_\Lambda \mathcal{X}}} \Omega(\text{pt}(\widehat{F}_\Lambda \mathcal{X})) = \Omega(\widehat{\top} \mathcal{X}).$$

By $\lambda_{\mathcal{U}\mathcal{X}}$ we actually mean the restriction of $\lambda_{\mathcal{U}\mathcal{X}}$ to $\Omega^n \mathcal{X} \subseteq \check{\mathbb{P}}(\mathcal{U}\mathcal{X})$. The map $k_{\widehat{F}_\Lambda \mathcal{X}}$ is the frame homomorphism given by $a \mapsto \{p \in \text{pt}(\widehat{F}_\Lambda \mathcal{X}) \mid p(a) = 1\}$. Then $\widehat{\Lambda} := \{\widehat{\lambda} \mid \lambda \in \Lambda\}$ is a geometric modal signature for $\widehat{\top}_\Lambda$.

► **Lemma 42.** *The assignment $\widehat{\lambda}$ is a natural transformation.*

Proof. For a continuous function $f : \mathcal{X} \rightarrow \mathcal{X}'$ the following diagram commutes in **Set**:

$$\begin{array}{ccccccc}
\Omega^n \mathcal{X}' & \xrightarrow{\lambda_{\mathcal{X}'}} & Z(\dot{F}_\Lambda \mathcal{X}') & \xrightarrow{Zq_{\mathcal{X}'}} & Z(\widehat{F}_\Lambda \mathcal{X}') & \xrightarrow{Zk_{\widehat{F}_\Lambda \mathcal{X}'}} & \Omega(\text{pt}(\widehat{F}_\Lambda \mathcal{X}')) \\
(f^{-1})^n \downarrow & & \downarrow (\mathbb{T}f)^{-1} & & \downarrow (\mathbb{T}f)^{-1} & & \downarrow \Omega(\text{pt}((\mathbb{T}f)^{-1})) \\
\Omega^n \mathcal{X} & \xrightarrow{\lambda_{\mathcal{X}}} & Z(\dot{F}_\Lambda \mathcal{X}) & \xrightarrow{Zq_{\mathcal{X}}} & Z(\widehat{F}_\Lambda \mathcal{X}) & \xrightarrow{Zk_{\widehat{F}_\Lambda \mathcal{X}}} & \Omega(\text{pt}(\widehat{F}_\Lambda \mathcal{X}))
\end{array}$$

Commutativity of the left square follows from naturality of λ , commutativity of the middle square follows from the proof of Lemma 36 and commutativity of the right square can be seen as follows: let $a'_1, \dots, a'_n \in \Omega \mathcal{X}'$, then

$$\begin{aligned}
& \Omega(\text{pt}((\mathbb{T}f)^{-1})) \circ Zk_{\widehat{F}_\Lambda \mathcal{X}'}(\lambda_{\mathcal{X}'}(a'_1, \dots, a'_n)) \\
&= \{q \in \text{pt}(\widehat{F}_\Lambda \mathcal{X}') \mid q \circ (\mathbb{T}f)^{-1}(\lambda_{\mathcal{X}'}(a'_1, \dots, a'_n)) = 1\} \\
&= Zk_{\widehat{F}_\Lambda \mathcal{X}'}((\mathbb{T}f)^{-1}(\lambda_{\mathcal{X}'}(a'_1, \dots, a'_n))).
\end{aligned}$$

So $\widehat{\lambda}$ is an open predicate lifting. ◀

The nature of the definitions of $\widehat{\mathbb{T}}_\Lambda$ and $\widehat{\Lambda}$ yields the following desirable results.

- **Proposition 43. 1.** *Let $\mathbb{T} : \mathbf{Set} \rightarrow \mathbf{Set}$ be a functor and Λ a collection of predicate liftings for \mathbb{T} . Then $\widehat{\Lambda}$ is characteristic for $\widehat{\mathbb{T}}_\Lambda$.*
2. *If $\lambda \in \Lambda$ are monotone, then $\widehat{\lambda} \in \widehat{\Lambda}$ is Scott-continuous.*

Proof. Let \mathcal{X} be a topological space. For the first item, we need to show that the collection

$$\{\widehat{\lambda}(a_1, \dots, a_n) \mid \lambda \in \Lambda \text{ } n\text{-ary}, a_i \in \Omega \mathcal{X}\} \quad (3)$$

forms a subbase for the topology on $\widehat{\mathbb{T}}_\Lambda \mathcal{X}$. An arbitrary nonempty open set of $\widehat{\mathbb{T}}_\Lambda \mathcal{X}$ is of the form $\tilde{x} = \{p \in \text{pt}(\widehat{F}_\Lambda \mathcal{X}) \mid p(x) = 1\}$, for $x \in \widehat{F}_\Lambda \mathcal{X}$. An arbitrary element of $\widehat{F}_\Lambda \mathcal{X}$ is the equivalence class of an arbitrary union of finite intersections of elements of the form $\lambda_{\mathcal{X}}(a_1, \dots, a_n)$, for $\lambda \in \Lambda$ and $a_1, \dots, a_n \in \Omega \mathcal{X}$. So we may write $x = \bigcup_{i \in I} (\bigcap_{j \in J_i} [\lambda_{\mathcal{X}}^{i,j}(a_1^{i,j}, \dots, a_{n_i}^{i,j})])$ for some index set I , finite index sets J_i , $\lambda^{i,j} \in \Lambda$ and open sets $a_k^{i,j} \in \Omega \mathcal{X}$. We get

$$\tilde{x} = \bigcup_{i \in I} \left(\bigcap_{j \in J_i} [\lambda_{\mathcal{X}}^{i,j}(a_1^{i,j}, \dots, a_{n_i}^{i,j})] \right) = \bigcup_{i \in I} \left(\bigcap_{j \in J_i} \widehat{\lambda}_{\mathcal{X}}^{i,j}(a_1^{i,j}, \dots, a_{n_i}^{i,j}) \right).$$

The second equality follows from Definition 41. This shows that the open sets in (3) indeed form a subbase for the open sets of $\widehat{\mathbb{T}}_\Lambda \mathcal{X}$.

The second item follows immediately from the definitions. ◀

6 Bisimulations

This section is devoted to bisimulations and bisimilarity between coalgebraic geometric models. We compare two notions of bisimilarity, modal equivalence (Definition 12) and behavioural equivalence (Definition 14). Again, where \mathbf{C} is be a full subcategory of **Top** and \mathbb{T} an endofunctor on \mathbf{C} , we give definitions and propositions in this generality where possible. When necessary, we restrict our scope to particular instances of \mathbf{C} .

► **Definition 44.** Let $\mathfrak{X} = (\mathcal{X}, \gamma, V)$ and $\mathfrak{X}' = (\mathcal{X}', \gamma', V')$ be two geometric \mathbb{T} -models. Let $B \subseteq \mathcal{X} \times \mathcal{X}'$ be a relation such that, equipped with the subspace topology, it is in \mathbf{C} and let $\pi : B \rightarrow \mathcal{X}$, $\pi' : B \rightarrow \mathcal{X}'$ be projections. Then B is called an *Aczel-Mendler bisimulation*

between \mathfrak{X} and \mathfrak{X}' if for all $(x, x') \in B$ we have $x \in V(p)$ iff $x' \in V'(p)$, and there exists a transition map $\beta : B \rightarrow \mathbb{T}B$ that makes π and π' coalgebra morphisms. Two states $x \in \mathbf{U}\mathcal{X}, x' \in \mathbf{U}\mathcal{X}'$ are called *bisimilar* if there is some Aczel-Mendler bisimulation linking them, notation $x \simeq x'$.

It follows from Proposition 13 that bisimilar states satisfy the same formulas. Furthermore, it easily follows by taking pushouts that Aczel-Mendler bisimilarity implies behavioural equivalence. If moreover \mathbb{T} preserves weak pullbacks, the converse holds as well [24].

However, we do not wish to make this assumption on topological spaces, since few functors seem to preserve weak pullbacks. For example, the Vietoris functor does not preserve weak pullbacks [5, Corollary 4.3] and neither does the monotone functor from Definition 17. (To see the latter statement, consider the example given in Section 4 of [13] and equip the sets in use with the discrete topology.) Therefore we define Λ -bisimulations for **Top**-coalgebras as an alternative to Aczel-Mendler bisimulations. This notion is an adaptation of ideas in [2, 10]. Under some conditions on Λ , Λ -bisimilarity coincides with behavioural equivalence.

In the next definition we need the concept of coherent pairs: If X and X' are two sets and $B \subseteq X \times X'$ is a relation, then a pair $(a, a') \in \mathbf{P}X \times \mathbf{P}X'$ is called *B-coherent* if $B[a] \subseteq a'$ and $B^{-1}[a'] \subseteq a$. For details and properties see section 2 in [14].

► **Definition 45.** Let \mathbb{T} be an endofunctor on \mathbf{C} , Λ a geometric modal signature for \mathbb{T} and $\mathfrak{X} = (\mathcal{X}, \gamma, V)$ and $\mathfrak{X}' = (\mathcal{X}', \gamma', V')$ two geometric \mathbb{T} -models. A Λ -bisimulation between \mathfrak{X} and \mathfrak{X}' is a relation $B \subseteq \mathbf{U}\mathcal{X} \times \mathbf{U}\mathcal{X}'$ such that for all $(x, x') \in B$, all $p \in \Phi$ and all tuples of B -coherent pairs of opens $(a_i, a'_i) \in \Omega\mathcal{X} \times \Omega\mathcal{X}'$ we have

$$x \in V(p) \quad \text{iff} \quad x' \in V'(p) \quad (4)$$

$$\gamma(x) \in \lambda_{\mathcal{X}}(a_1, \dots, a_n) \quad \text{iff} \quad \gamma'(x') \in \lambda_{\mathcal{X}'}(a'_1, \dots, a'_n). \quad (5)$$

Two states are called Λ -bisimilar if there is a Λ -bisimulation linking them, notation: $x \simeq_{\Lambda} x'$.

We give an alternative characterisation of (5) to elucidate the connection with [2].

► **Remark 46.** Let $B \subseteq \mathcal{X} \times \mathcal{X}'$ be a relation endowed with the subspace topology and let $\pi : B \rightarrow \mathcal{X}$ and $\pi' : B \rightarrow \mathcal{X}'$ be projections. Then $(a, a') \in \Omega\mathcal{X} \times \Omega\mathcal{X}'$ is B -coherent iff $\pi^{-1}(a) = (\pi')^{-1}(a')$.

Let P be the pullback of the cospan $\Omega\mathcal{X} \xrightarrow{\Omega\pi} \Omega B \xleftarrow{\Omega\pi'} \Omega\mathcal{X}'$ in \mathbf{Frm} and let $p : P \rightarrow \mathcal{X}$ and $p' : P \rightarrow \mathcal{X}'$ be the corresponding projections. Then the B -coherent pairs are precisely $(p(x), p'(x))$, where x ranges over P . It follows from the definitions that equation (5) holds for all B -coherent pairs if and only if

$$\Omega\pi \circ \Omega\gamma \circ \lambda_{\mathcal{X}} \circ p^n = \Omega\pi' \circ \Omega\gamma' \circ \lambda_{\mathcal{X}'} \circ (p')^n,$$

where λ is n -ary.

As desired, Λ -bisimilar states satisfy the same formulas.

► **Proposition 47.** Let \mathbb{T} be an endofunctor on \mathbf{C} and Λ a geometric modal signature for \mathbb{T} . Then $\simeq_{\Lambda} \subseteq \equiv_{\Lambda}$.

Proof. Let B be a Λ -bisimulation between geometric \mathbb{T} -models \mathfrak{X} and \mathfrak{X}' , and suppose $x B x'$. Using induction on the complexity of the formula, we show that $\mathfrak{X}, x \Vdash \varphi$ iff $\mathfrak{X}', x' \Vdash \varphi$ for all $\varphi \in \text{GML}(\Lambda)$. The propositional case is by definition, and \wedge and \vee are routine. Suppose $\mathfrak{X}, x \Vdash \heartsuit^{\lambda}(\varphi_1, \dots, \varphi_n)$, then $\gamma(x) \in \lambda_{\mathcal{X}}(\llbracket \varphi_1 \rrbracket^{\mathfrak{X}}, \dots, \llbracket \varphi_n \rrbracket^{\mathfrak{X}})$. By the induction hypothesis $(\llbracket \varphi_i \rrbracket^{\mathfrak{X}}, \llbracket \varphi_i \rrbracket^{\mathfrak{X}'})$ is B -coherent for all i . Then $\gamma'(x') \in \lambda_{\mathcal{X}'}(\llbracket \varphi_1 \rrbracket^{\mathfrak{X}'}, \dots, \llbracket \varphi_n \rrbracket^{\mathfrak{X}'})$ since B is a Λ -bisimulation, hence $\mathfrak{X}', x' \Vdash \heartsuit^{\lambda}(\varphi_1, \dots, \varphi_n)$. The converse is proven symmetrically. ◀

The proof of the next proposition is similar to [2, Proposition 20].

► **Proposition 48.** *Let T be an endofunctor on \mathbf{C} and Λ a geometric modal signature for T . Then $\Leftrightarrow \subseteq \Leftrightarrow_{\Lambda}$.*

We know by now that Λ -bisimilarity implies modal equivalence. Furthermore, if T is an endofunctor on \mathbf{Top} which preserves sobriety, modal equivalence implies behavioural equivalence. In order to prove a converse, i.e. that behavioural equivalence implies Λ -bisimilarity, we need to assume that the geometric modal signature is strong.

Recall that two elements x, x' in two models are behaviourally equivalent in $\mathbf{Mod}(\mathsf{T})$, notation: $\approx_{\mathbf{Mod}(\mathsf{T})}$, if there exist morphisms f, f' in $\mathbf{Mod}(\mathsf{T})$ such that $f(x) = f'(x')$.

► **Proposition 49.** *Let Λ a strongly monotone geometric modal signature for $\mathsf{T} : \mathbf{C} \rightarrow \mathbf{C}$ and let $\mathfrak{X} = (\mathcal{X}, \gamma, V)$ and $\mathfrak{X}' = (\mathcal{X}', \gamma', V')$ be two geometric T -models. Then $\approx_{\mathbf{Mod}(\mathsf{T})} \subseteq \Leftrightarrow_{\Lambda}$.*

Proof. Suppose x and x' are behaviourally equivalent. Then there are some geometric T -model $\mathfrak{Y} = (\mathcal{Y}, \nu, V_{\mathcal{Y}})$ and T -model morphisms $f : \mathfrak{X} \rightarrow \mathfrak{Y}$ and $f' : \mathfrak{X}' \rightarrow \mathfrak{Y}$ such that $f(x) = f'(x')$. We will show that

$$B = \{(u, u') \in X \times X' \mid f(u) = f'(u')\}. \quad (6)$$

is a Λ -bisimulation B linking x and x' .

Clearly $x B x'$. It follows from Proposition 13 that u and u' satisfy precisely the same formulas whenever $(u, u') \in B$. Suppose $\lambda \in \Lambda$ is n -ary and for $1 \leq i \leq n$ let (a_i, a'_i) be a B -coherent pair of opens. Suppose $u B u'$ and $\gamma(u) \in \lambda_{\mathcal{X}}(a_1, \dots, a_n)$. We will show that $\gamma'(u') \in \lambda_{\mathcal{X}'}(a'_1, \dots, a'_n)$. The converse direction is similar. By monotonicity and naturality of λ we obtain

$$\gamma(u) \in \lambda_{\mathcal{X}}(a_1, \dots, a_n) \subseteq \lambda_{\mathcal{X}}(f^{-1}(f[a_1]), \dots, f^{-1}(f[a_n])) = (\mathsf{T}f)^{-1}(\lambda_{\mathcal{Y}}(f[a_1], \dots, f[a_n])),$$

so $(\mathsf{T}f)(\gamma(u)) \in \lambda_{\mathcal{Y}}(f[a_1], \dots, f[a_n])$. (The $f[a_i]$ need not be open in \mathcal{Y} , but since λ is strong, $\lambda_{\mathcal{Y}}(f[a_1], \dots, f[a_n])$ is defined.) Because f and f' are coalgebra morphisms and $f(u) = f'(u')$ we have $(\mathsf{T}f)(\gamma(u)) = \nu(f(u)) = \nu(f'(u')) = (\mathsf{T}f')(\gamma'(u'))$. Finally, we get

$$\begin{aligned} \gamma'(u') &\in (\mathsf{T}f')^{-1}(\lambda_{\mathcal{Y}}(f[a_1], \dots, f[a_n])) \\ &= \lambda_{\mathcal{X}'}((f')^{-1}(f[a_1]), \dots, (f')^{-1}(f[a_n])) && \text{(naturality of } \lambda) \\ &= \lambda_{\mathcal{X}'}(B[a_1], \dots, B[a_n]) && \text{(strong monotonicity of } \lambda) \\ &\subseteq \lambda_{\mathcal{X}'}(a'_1, \dots, a'_n). && \text{(coherence of } (a_i, a'_i)) \quad \blacktriangleleft \end{aligned}$$

► **Remark 50.** If $\mathbf{C} = \mathbf{KHaus}$ in the proposition above, then Proposition 22 allows us to drop the assumption that Λ be strong.

Let T be an endofunctor on \mathbf{Top} and let Λ be a geometric modal signature for T . The following diagram summarises the results from Propositions 47 and 49 and Theorem 31. The arrows indicate that one form of equivalence implies the other. Here (1) holds if T preserves weak pullbacks, (2) is true when Λ is Scott-continuous and characteristic and T preserves sobriety, and (3) holds when Λ is strongly monotone. Note that the converse of (2) always holds, because morphisms preserve truth (Proposition 13).

$$\begin{array}{ccc} & \xrightarrow{\quad (1) \quad} & \\ \Leftrightarrow & \xrightarrow{\quad} \Leftrightarrow_{\Lambda} \xrightarrow{\quad} \equiv_{\Lambda} \xrightarrow{\quad (2) \quad} \approx_{\mathbf{Mod}(\mathsf{T})} & \\ & \xleftarrow{\quad (3) \quad} & \end{array} \quad (7)$$

As stated in the introduction we are not only interested in endofunctors on **Top**, but also in endofunctors on full subcategories of **Top**, in particular **KHaus**.

The implications in the diagram hold for endofunctors on **Sob** as well (use Remark 32). Moreover, with some extra effort it can be made to work for endofunctors on **KSob** as well. In order to achieve this, we have to redo the proof for the bi-implication between modal equivalence and behavioural equivalence. This is the content of the following theorem.

► **Theorem 51.** *Let T be an endofunctor on **KSob**, Λ a Scott-continuous characteristic geometric modal signature for T and $\mathfrak{X} = (\mathcal{X}, \gamma, V)$ and $\mathfrak{X}' = (\mathcal{X}', \gamma', V')$ two geometric T -models. Then $\equiv_{\Lambda} = \simeq_{\mathbf{Mod}(\mathsf{T})}$.*

Proof. If x and x' are behaviourally equivalent then they are modally equivalent by Proposition 13. The converse direction can be proved using similar reasoning as in Section 4. The major difference is the following: We define an equivalence relation \equiv_2 on $\mathbf{GML}(\Lambda)$ by $\varphi \equiv_2 \psi$ iff $\llbracket \varphi \rrbracket^{\mathfrak{X}} = \llbracket \psi \rrbracket^{\mathfrak{X}}$ and $\llbracket \varphi \rrbracket^{\mathfrak{X}'} = \llbracket \psi \rrbracket^{\mathfrak{X}'}$. (Note that \mathfrak{X} and \mathfrak{X}' are now fixed.) That is, $\varphi \equiv_2 \psi$ iff φ and ψ are satisfied by precisely the same states in \mathfrak{X} and \mathfrak{X}' (compare Definition 23). The frame $\mathbf{E}_2 := \mathbf{GML}(\Lambda)/\equiv_2$ can then be shown to be a compact frame and hence $\mathcal{Z}_2 := \mathbf{ptE}_2$ is a compact sober space. The remainder of the proof is analogous to the proof of Theorem 31. A detailed proof can be found in [11, Theorem 3.34]. ◀

We summarise the results for **Top** and two of its full subcategories:

► **Theorem 52.** *Let T be an endofunctor on **Top**, **Sob** or **KSob** and Λ a Scott-continuous characteristic strongly monotone geometric modal signature for T . If x and x' are two states in two geometric T -models, then*

$$x \simeq_{\Lambda} x' \quad \text{iff} \quad x \equiv_{\Lambda} x' \quad \text{iff} \quad x \simeq_{\mathbf{Mod}(\mathsf{T})} x'.$$

For coalgebras over base category **KHaus** we have:

► **Theorem 53.** *Let T be an endofunctor on **KHaus** which is the restriction of an endofunctor S on **Sob** or **KSob** and let Λ be a Scott-continuous characteristic monotone geometric modal signature for S (hence for T). Then $x \simeq_{\Lambda} x'$ iff $x \equiv_{\Lambda} x'$.*

Both the Vietoris functor $\mathbf{V}_{\mathbf{kh}}$ and the monotone functor $\mathbf{D}_{\mathbf{kh}}$, together with their respective open predicate liftings for box and diamond, satisfy the premises of this theorem.

7 Conclusion

We have started building a framework for coalgebraic *geometric* logic and we have investigated examples of concrete functors. There are still many unanswered and interesting questions. We outline possible directions for further research.

Modal equivalence versus behavioural equivalence From Theorem 52 we know that modal equivalence and behavioural equivalence coincide in $\mathbf{Mod}(\mathsf{T})$ if T is an endofunctor on **KSob**, **Sob** or an endofunctor on **Top** which preserves sobriety. A natural question is whether the same holds when T is an endofunctor on **KHaus**.

When does a lifted functor restrict to KHaus? We know of two examples, namely the powerset functor with the box and diamond lifting, and the monotone functor on **Set** with the box and diamond lifting, where the lifted functor on **Top** restricts to **KHaus**. It would be interesting to investigate whether there are explicit conditions guaranteeing that the lift of a functor restricts to **KHaus**. These conditions could be either for the **Set**-functor one starts with, or the collection of predicate liftings for this functor, or both.

Bisimulations In [2] the authors define Λ -bisimulations (which are inspired by [10]) between **Set**-coalgebras. In this paper we define Λ -bisimulations between **C**-coalgebras. A similar definition yields a notion of Λ -bisimulation between **Stone**-coalgebras, where the interpretants of the proposition letters are clopen sets, see [11, Definition 2.19]. This raises the question whether a more uniform treatment of Λ -bisimulations is possible, which encompasses all these cases.

Modalities and finite observations Geometric logic is generally introduced as the logic of finite observations, and this explains the choice of connectives (\wedge , \vee and, in the first-order version, \exists). We would like to understand to which degree *modalities* can safely be added to the base language, without violating the (semantic) intuition of finite observability. Clearly there is a connection with the requirement of *Scott-continuity* (preservation of directed joins), and we would like to make this connection precise, specifically in the topological setting.

References

- 1 S. Abramsky. *Domain Theory and the Logic of Observable Properties*. PhD thesis, University of London, 1987.
- 2 Z. Bakhtiari and H.H. Hansen. Bisimulation for Weakly Expressive Coalgebraic Modal Logics. In *7th Conference on Algebra and Coalgebra in Computer Science, CALCO 2017, June 12-16, 2017, Ljubljana, Slovenia*, pages 4:1–4:16, 2017. doi:10.4230/LIPIcs.CALCO.2017.4.
- 3 G. Bezhanishvili, N. Bezhanishvili, and J. Harding. Modal compact Hausdorff spaces. *Journal of Logic and Computation*, 25(1):1–35, 2015. doi:10.1093/logcom/exs030.
- 4 N. Bezhanishvili, J. de Groot, and Y. Venema. Coalgebraic Geometric Logic, 2019. arXiv:1903.08837.
- 5 N. Bezhanishvili, G. Fontaine, and Y. Venema. Vietoris bisimulations. *Journal of Logic and Computation*, 20(5):1017–1040, 2010.
- 6 B.F. Chellas. *Modal Logic: An Introduction*. Cambridge University Press, 1980.
- 7 L.-T. Chen and A. Jung. On a Categorical Framework for Coalgebraic Modal Logic. *Electronic Notes in Theoretical Computer Science*, 308:109–128, 2014.
- 8 C. Cirstea, A. Kurz, D. Pattinson, L. Schröder, and Y. Venema. Modal logics are coalgebraic. In S. Abramsky, E. Gelenbe, and V. Sassone, editors, *Visions of Computer Science, BCS International Academic Research Conference (BCS 2008)*, pages 129–140. British Computer Society, 2008.
- 9 S. Enqvist and S. Sourabh. Bisimulations for coalgebras on Stone spaces. *Journal of Logic and Computation*, 28(6):991–1010, 2018. doi:10.1093/logcom/exy001.
- 10 D. Gorín and L. Schröder. Simulations and Bisimulations For Coalgebraic Modal Logics. In R. Heckel and S. Milius, editors, *5th Conference on Algebra and Coalgebra in Computer Science, CALCO 2013*, pages 253–266. Springer, 2013.
- 11 J. de Groot. Coalgebraic geometric logic. Master’s thesis, University of Amsterdam, available at <https://esc.fnwi.uva.nl/thesis/centraal/files/f2119048545.pdf>.
- 12 H.H. Hansen. Monotonic modal logics, 2003. Master’s thesis, Institute for Logic, Language and Computation, University of Amsterdam.
- 13 H.H. Hansen and C. Kupke. A coalgebraic Perspective on Monotone Modal Logic. *Electronic Notes in Theoretical Computer Science*, 106:121–143, 2004.
- 14 H.H. Hansen, C. Kupke, and E. Pacuit. Neighbourhood structures: bisimilarity and basic model theory. *Logical Methods in Computer Science*, 5(2), April 2009.
- 15 P.T. Johnstone. *Stone Spaces*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1982.
- 16 P.T. Johnstone. *Sketches of an Elephant: A Topos Theory Compendium*. Number v. 2 in Oxford Logic Guides. Clarendon Press, 2002.

- 17 C. Kupke, A. Kurz, and D. Pattinson. Algebraic Semantics for Coalgebraic Logics. *Electronic Notes in Theoretical Computer Science*, 106:219–241, 2004. Proceedings of the Workshop on Coalgebraic Methods in Computer Science (CMCS).
- 18 C. Kupke, A. Kurz, and D. Pattinson. Ultrafilter Extensions for Coalgebras. In J. Luiz Fiadeiro, N. Harman, M. Roggenbach, and J. Rutten, editors, *Algebra and Coalgebra in Computer Science*, pages 263–277, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
- 19 C. Kupke, A. Kurz, and Y. Venema. Stone coalgebras. *Theoretical Computer Science*, 327(1):109–134, 2004. Selected Papers of CMCS '03. doi:10.1016/j.tcs.2004.07.023.
- 20 C. Kupke and D. Pattinson. Coalgebraic semantics of modal logics: An overview. *Theoretical Computer Science*, 412(38):5070–5094, 2011. CMCS Tenth Anniversary Meeting.
- 21 E. Michael. Topologies on spaces of subsets. *Transactions of the American Mathematical Society*, 71:152–182, 1951.
- 22 L.S. Moss. Coalgebraic logic. *Annals of Pure and Applied Logic*, 96(1):277–317, 1999. doi:10.1016/S0168-0072(98)00042-6.
- 23 D. Pattinson. Coalgebraic modal logic: soundness, completeness and decidability of local consequence. *Theoretical Computer Science*, 309(1):177–193, 2003.
- 24 J.J.M.M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249((1)):3–80, 2000.
- 25 L. Schröder. Expressivity of Coalgebraic Modal Logic: The Limits and Beyond. In V. Sassone, editor, *Foundations of Software Science and Computational Structures*, pages 440–454, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
- 26 Y. Venema. Coalgebra and Modal Logic, 2017. Course notes. ILLC, UvA. URL: <https://staff.science.uva.nl/y.venema/teaching/ml/notes/cml-2017.pdf>.
- 27 Y. Venema, S. Vickers, and J. Vosmaer. Generalized powerlocales via relation lifting. *CoRR*, abs/1202.3264, 2012.
- 28 S.J. Vickers. *Topology via Logic*. Cambridge University Press, New York, NY, USA, 1989.

Coinduction in Flow: The Later Modality in Fibrations

Henning Basold 

CNRS, ENS de Lyon, France

LIACS – Leiden University, The Netherlands

h.basold@liacs.leidenuniv.nl

Abstract

This paper provides a construction on fibrations that gives access to the so-called later modality, which allows for a controlled form of recursion in coinductive proofs and programs. The construction is essentially a generalisation of the topos of trees from the codomain fibration over sets to arbitrary fibrations. As a result, we obtain a framework that allows the addition of a recursion principle for coinduction to rather arbitrary logics and programming languages. The main interest of using recursion is that it allows one to write proofs and programs in a goal-oriented fashion. This enables easily understandable coinductive proofs and programs, and fosters automatic proof search.

Part of the framework are also various results that enable a wide range of applications: transportation of (co)limits, exponentials, fibred adjunctions and first-order connectives from the initial fibration to the one constructed through the framework. This means that the framework extends any first-order logic with the later modality. Moreover, we obtain soundness and completeness results, and can use up-to techniques as proof rules. Since the construction works for a wide variety of fibrations, we will be able to use the recursion offered by the later modality in various context. For instance, we will show how recursive proofs can be obtained for arbitrary (syntactic) first-order logics, for coinductive set-predicates, and for the probabilistic modal μ -calculus. Finally, we use the same construction to obtain a novel language for probabilistic productive coinductive programming. These examples demonstrate the flexibility of the framework and its accompanying results.

2012 ACM Subject Classification Theory of computation \rightarrow Logic

Keywords and phrases Coinduction, Fibrations, Later Modality, Recursive Proofs, Up-to techniques, Probabilistic Logic, Probabilistic Programming

Digital Object Identifier 10.4230/LIPIcs.CALCO.2019.8

Related Version <https://arxiv.org/abs/1802.07143>

Funding *Henning Basold*: This work has been funded by the European Research Council (ERC) under the EU’s Horizon 2020 programme (CoVeCe, grant agreement No 678157), and was supported by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program “Investissements d’Avenir” (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

1 Introduction

Recursion is one of the most fundamental notions in computer science and mathematics, be it as the foundation of computability, or to define and reason about structures determined by repeated constructions. In this paper, we will focus on the use of recursion as a method for coinductive proofs and coinductive programming.

Usually, coinductive programming is presented by means of coiteration schemes and coinduction as bisimulation proof principle. Coiteration schemes are a syntactic implementation of coalgebras and their coinductive extension to a homomorphism into the final coalgebra [32, 48]. The bisimulation proof principle, on the other hand, asserts that bisimilarity implies equality in the final coalgebra [29, 36, 60]. There are, however, also different approaches that break with this dogma. In coinductive programming, guarded recursion [5, 6, 16, 50, 52], and sets of recursive equations [1, 33, 61] have been used to construct elements of final coalgebras and of coinductive types. On the side of proofs and semantics, several improvements of coinduction



© Henning Basold;

licensed under Creative Commons License CC-BY

8th Conference on Algebra and Coalgebra in Computer Science (CALCO 2019).

Editors: Markus Roggenbach and Ana Sokolova; Article No. 8; pp. 8:1–8:22

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

have been suggested: simplification of invariants [63] via up-to techniques [19, 54, 58] and the companion [11, 55, 56], incremental techniques [38, 51], games [53, 65], and basic cyclic proofs for stream equality [57]. In this paper, we will focus on guarded recursion because it can be widely applied, and because it leads to clean proof and programming methods.

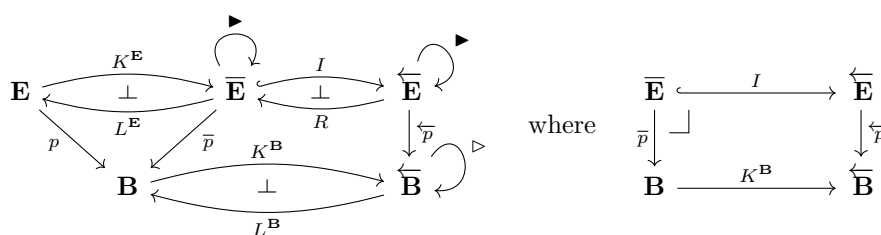
A concrete appearance of coinduction can be found in the modal μ -calculus $L\mu$ [46, 20] and its quantitative interpretations [39] $pL\mu$ or $L\mu$ [49] in form of Park’s rule, which assert that if $\psi \rightarrow \varphi[\psi/X]$, then $\psi \rightarrow \nu X.\varphi$. This rule says that an implication with a greatest fixed point as conclusion can be proven by showing that ψ is an *invariant* for φ . Kozen [46] gave an axiomatisation of $L\mu$ based on this rule, and its dual, that turned out to be complete [75]. Thus, this axiomatisation is expressive, but often difficult to use in practice, let alone for proof search. It should be noted that $L\mu$ is decidable if it is interpreted in classical logic. The goal of this work is, however, to develop techniques that can also be used to obtain (constructive) proof objects and can be applied to more general logics. Thus, our focus will be on improving the axiomatisation of $L\mu$ and of coinductive proofs in general.

Coming back to Park’s rule, we often find ourselves having to prove $\psi \rightarrow \nu X.\varphi$ for a formula ψ , which is not an invariant. We are then required to find an invariant ψ' , such that, $\psi \rightarrow \psi'$. Finding such an invariant can be difficult in general and it does not fit common practice. Instead, it would be preferable if we could incrementally construct the proof for $\psi \rightarrow \nu X.\varphi$ rather than guessing an invariant ψ' . Such an incremental construction leads to a recursive proof methodology for coinductive proofs. As such incremental methods are valuable in any theory that is based on coiteration or coinduction schemes, we set out in this paper to replace invariant guessing by a general iterative programming and proof method.

The proposed iterative method will be given in form of a framework that introduces recursion into coinductive proofs and programs, while preserving soundness and termination. This framework is centred around the so-called later modality [52], which allows for us to control the use of recursion and thereby avoid the introduction of non-termination and inconsistencies. The later modality has been successfully used in the context of semantics [16, 72], programming [5, 6, 50], and reasoning [23, 14]. Ultimately, we generalise the work of Birkedal et al. [16] on the topos of trees to arbitrary fibrations with the effect of much wider applicability to, for example, quantitative reasoning and probabilistic programming.

In the case of $L\mu$, we extend the logic with the later modality as a new logical connective. Given a formula φ , we thus obtain a formula $\blacktriangleright\varphi$. This formula should be read as “later φ ”, which allows us to formulate that knowledge varies over time. The later modality comes with three crucial axioms: $\varphi \rightarrow \blacktriangleright\varphi$ (next), $\blacktriangleright(\varphi \rightarrow \psi) \rightarrow \blacktriangleright\varphi \rightarrow \blacktriangleright\psi$ (monotonicity), and $(\blacktriangleright\varphi \rightarrow \varphi) \rightarrow \varphi$ (fixed point or Löb). It is the Löb rule that introduces recursion into the logic, and it should be read as “if we can prove φ from the assumption that φ holds later, then φ holds at any time”. However, the assumption $\blacktriangleright\varphi$ introduced by the Löb rule cannot be used directly. We need one final axiom for that: $\varphi[\blacktriangleright\nu X.\varphi/X] \rightarrow \nu X.\varphi$ (step). These axioms can be combined to obtain recursive proofs, as we will show later. As an appetiser, the reader may have a look already at Figure 3 on Page 15.

The reader may have noticed that the first three axioms, next, monotonicity and Löb, are independent of the logic at hand. Only the step axiom makes use of the structure of formulas. This observation is what enables the topos of trees and the framework presented here to work. More precisely, we will start with a given fibration $p: \mathbf{E} \rightarrow \mathbf{B}$ and construct a new fibration $\overleftarrow{p}: \overleftarrow{\mathbf{E}} \rightarrow \overleftarrow{\mathbf{B}}$ out of it. This fibration will have, under mild conditions, the later modality as a map of fibrations $(\blacktriangleright, \blacktriangleright)$ on it. The next and Löb axioms correspond then to certain morphisms in $\overleftarrow{\mathbf{E}}$, while monotonicity says that \blacktriangleright is a strong functor. From a logical perspective, it is more natural to consider another fibration $\overline{p}: \overline{\mathbf{E}} \rightarrow \mathbf{B}$ over the same base category as the initial fibration. In this fibration, we will not only have access to the later modality and its axioms, but also to quantifiers that are present in the original fibration p .



■ **Figure 1** Relation between p (base logic), \overleftarrow{p} (all chains in p) and \bar{p} (chains with constant index).

Contributions. Apart from the applications to the probabilistic modal μ -calculus and to probabilistic programming, the technical contributions of this paper are as follows. Given a fibration p and a well-ordered class \mathbf{I} , we let $\bar{\mathbf{E}}$ be the category of \mathbf{I}^{op} -indexed chains in \mathbf{E} , that is, functors $\sigma: \mathbf{I}^{\text{op}} \rightarrow \mathbf{E}$. The fibration \overleftarrow{p} is given by post-composition with p and thus maps a chain to the chain of its indices given by p . On this fibration, we construct the later modality and find all its good properties. We then restrict our attention to the fibration $\bar{p}: \bar{\mathbf{E}} \rightarrow \mathbf{B}$, which consists only of chains with constant index. In other words, \bar{p} is given by the change-of-base (pullback) along the functor $K^{\mathbf{B}}: \mathbf{B} \rightarrow \overleftarrow{\mathbf{B}}$ that maps $I \in \mathbf{B}$ to the chain that is equal to I at every position. This is indicated in the right diagram in Figure 1. The diagram on the left summarises the most important ingredients of the framework:

- the later modality is a map of fibrations $\blacktriangleright: \bar{p} \rightarrow \overleftarrow{p}$ and $(\blacktriangleright, \blacktriangleright): \overleftarrow{p} \rightarrow \overleftarrow{p}$ with a natural transformation $\text{next}: \text{Id} \Rightarrow \blacktriangleright$ (Theorem 16 and Theorem 17);
- \overleftarrow{p} and \bar{p} are fibred Cartesian closed categories and feature the Löb rule as morphism $\text{löb}_\sigma: \sigma \blacktriangleright^\sigma \rightarrow \sigma$ that fulfils a unique solution condition (Theorem 20 and Theorem 27);
- fixed points of so-called locally contractive functors on \overleftarrow{p} and \bar{p} (Theorem 31);
- the final chain construction of final coalgebras via a locally contractive functor (Theorem 34) and up-to techniques as proof rules (Theorem 35);
- if \mathbf{B} has \mathbf{I}^{op} -limits, then there is an adjunction $K^{\mathbf{B}} \dashv L^{\mathbf{B}}$ between \mathbf{B} and $\overleftarrow{\mathbf{B}}$, and an adjunction $I \dashv R$ between $\bar{\mathbf{E}}$ and $\overleftarrow{\mathbf{E}}$ (Theorem 19);
- fibred \mathbf{I}^{op} -limits in p give a fibred adjunction $K^{\mathbf{E}} \dashv L^{\mathbf{E}}$ between \mathbf{E} and $\bar{\mathbf{E}}$ (Theorem 19);
- if p is a first-order fibration, then \bar{p} is a first-order fibration and $L^{\mathbf{E}}$ preserves truth of first-order formulas if disjunction, existentials and equality preserve \mathbf{I}^{op} -limits (Theorem 41).

Particularly interesting is that \bar{p} is a first-order fibration, in other words, models first-order logic. This result can be restricted to any subset of connectives, which allows us to extend any logic with the later modality and its axioms. The adjunction between p and \bar{p} shows then that this yields a sound and complete axiomatisation of coinductive predicates. We leverage this generality to devise a novel proof system for the probabilistic modal μ -calculus and a language for productive probabilistic programming with coinductive types.

Another interesting aspect of the diagram is that one of the central results used by Hasuo et al. [35] (Lem. 3.5) appears here as the composition $L^{\mathbf{E}} \circ R: \overleftarrow{\mathbf{E}} \rightarrow \mathbf{E}$. In fact, the results in [35] tell us under which conditions we can use the finite ordinals ω as index \mathbf{I} to obtain a sound and complete proof system for coinductive predicates.

Organisation. The framework is introduced in the following steps. First, we provide in Section 2 a brief overview over fibrations, coinductive predicates and well-founded induction. Next, we describe in Section 3 the chain fibrations \overleftarrow{p} and \bar{p} , construct the later modality and give some basic results. Section 4 is devoted to show that functor fibrations are fibred Cartesian closed and to the Löb rule. In Section 5 we construct fixed points of so-called

locally contractive functors, both, on the whole fibration and on the fibres. Moreover, we show how the final chain arises as locally contractive functor, and how this leads to the proof rule “step” that we saw above. This allows us also to obtain proof rules on the final chain for compatible up-to techniques. As promised, we prove in Section 6 that \bar{p} is a first-order fibration. Furthermore, we give the adjunctions from Figure 1 that relate the various fibrations. The flexibility of the framework is then demonstrated by providing a recursive proof system for probabilistic $L\mu$ and a language for guarded recursive probabilistic programming in Section 7. We conclude with a few remarks and future work in Section 8.

Related Work. To a large part, the present paper generalises the work of Birkedal et al. [16] from the codomain fibration $\mathbf{Set}^{\rightarrow} \rightarrow \mathbf{Set}$ of sets to arbitrary fibrations. That [16] was so restrictive is not so surprising, as the intention there was to construct models of programming languages, rather than applying the developed techniques to proofs. Going beyond the category of sets also means that one has to involve much more complicated machinery to obtain exponential objects, see Section 4. Later, Bizjak et al. [17] extended the techniques from [16] to dependent type theory, thereby enabling reasoning by means of recursive proofs in a syntactic type theory. However, also this is a very specific setting, which rules out the main examples that we are interested in here. Similarly, also the parameterised coinduction in categories [51] and in lattices [38] is too restrictive, as they only apply to, respectively, propositional and to set-theoretic settings. It might be possible to develop parameterised coinduction in the setting of fibrations by using the companion [11, 55, 56], but we leave this question for another time. Recursion is also central to cyclic proof systems [21, 24, 26, 64]. These are particularly useful in settings that require proofs by induction or coinduction because cyclic proof systems ease proofs enormously compared to the invariant-based method of (co)induction schemes. Nothing comes for free though: In this case checking proofs becomes more difficult, as the correctness conditions are typically global for a proof tree and not compositional. For the same reason, also soundness proofs are often rather complex. The framework we study here gives rise to proof rules that require no further global condition on proofs, which straightforwardly yields proof checking [8] and soundness. Higher-order recursion has also been studied in other categorical settings like topos theory [47, 40] or monoidal categories [30, 34]. Unfortunately, these neither apply to our examples of interest, nor do they provide the logical results and constructions that appear in this paper.

Finally, in the realm of algorithmic proofs, circular proofs have been used to automatically prove identities of streams [57]. Else, computer-supported coinduction is usually limited to proof checking [31, 18, 25]. There have been limited approaches to combine coinduction with resolution [66]. In [10], we were able to go beyond the state of the art by extending uniform proofs to coinduction and using the framework presented in this paper as logical foundation. This shows that the framework of this paper paves the way for algorithmic proof search.

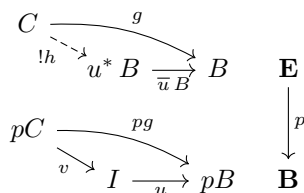
2 Preliminaries

We begin by recalling the terminology of fibrations, coalgebras, coinductive predicates, and well-founded induction. Moreover, we discuss examples that underlie the motivating applications of this paper.

2.1 Fibrations

One of the central notions used in this paper are fibrations [13, 41, 71], as they are an elegant way of capturing (typed) variables in a (higher-order) predicate logic.

► **Definition 1.** Let $p : \mathbf{E} \rightarrow \mathbf{B}$ be a functor, where \mathbf{E} is called the total category and \mathbf{B} the base category. A morphism $f : A \rightarrow B$ in \mathbf{E} is said to be Cartesian over $u : I \rightarrow J$, provided that i) $pf = u$, and ii) for all $g : C \rightarrow B$ in \mathbf{E} and $v : pC \rightarrow I$ with $pg = u \circ v$ there is a unique $h : C \rightarrow A$ such that $f \circ h = g$. For p to be a fibration, we require that for every $B \in \mathbf{E}$ and $u : I \rightarrow pB$ in \mathbf{B} , there is a cartesian morphism $f : A \rightarrow B$ over u . Finally, a fibration is cloven, if it comes with a unique choice for A and f , in which case we denote A by $u^* B$ and f by $\bar{u} B$, as displayed in the diagram in Figure 2. ◻



■ **Figure 2** Cartesian Lifting in a Fibration p .

On cloven fibrations, we can define for each $u : I \rightarrow J$ in \mathbf{B} a functor, the *reindexing along u* , as follows. Let us denote by \mathbf{E}_I the category having objects A with $p(A) = I$ and morphisms $f : A \rightarrow B$ with $p(f) = \text{id}_I$. We call \mathbf{E}_I the *fibre above I* and the morphisms in \mathbf{E}_I *vertical*. The assignment of $u^* B$ to B for a cloven fibration can then be extended to a functor $u^* : \mathbf{E}_J \rightarrow \mathbf{E}_I$. Moreover, one can show that there are natural isomorphisms $\text{id}_I^* \cong \text{Id}_{\mathbf{E}_I}$ and $(v \circ u)^* \cong u^* \circ v^*$ subject to some coherence conditions.

For a fibration $p : \mathbf{E} \rightarrow \mathbf{B}$ and a functor $F : \mathbf{C} \rightarrow \mathbf{B}$, we can form a new fibration $F^*(p) : F^*(\mathbf{E}) \rightarrow \mathbf{C}$ by pulling p back along F , see [41]. The fibration $F^*(p)$ is said to be obtained by *change-of-base*. Given another fibration $q : \mathbf{D} \rightarrow \mathbf{A}$, a *map of fibrations* $p \rightarrow q$ is a pair (F, G) of functors $F : \mathbf{A} \rightarrow \mathbf{B}$ and $G : \mathbf{D} \rightarrow \mathbf{E}$, with $p \circ G = F \circ q$ and such that G preserves Cartesian morphisms. This means in particular for $u : I \rightarrow J$ and $A \in \mathbf{E}_J$ that for $G(u^* A) \cong (Fu)^*(GA)$. Finally, the fibration p is said to have *fibred \diamond* (certain limits, colimits, exponentials, etc.), if every fibre has \diamond and reindexing preserves these.

Let \mathbf{C} be a Cartesian closed category. We denote for $f : Y \rightarrow X$ by $\ulcorner f \urcorner : \mathbf{1} \rightarrow X^Y$ the code of f . Recall [45] that a functor $F : \mathbf{C} \rightarrow \mathbf{C}$ is *strong* if there is natural family of morphisms $\text{st}_{X,Y}^F : X^Y \rightarrow FX^{FY}$, s.t. $\text{st}_{X,Y}^F \circ \ulcorner f \urcorner = \ulcorner Ff \urcorner$. A map of fibrations $(F, G) : p \rightarrow p$ is strong if both F and G are strong, and $p \text{st}^G = \text{st}^F$.

As the definition of fibrations and the associated notions are fairly abstract, let us give a few examples. There are five examples that we shall use to illustrate different aspects of the theory: predicates over sets, quantitative predicates, syntactic logic, the codomain fibration over the category of sets, and categories as trivial fibrations. The codomain fibration will allow us to recover later the topos of trees. We begin with the simplest example, namely that of predicates. Despite its simplicity, it is already quite useful because it allows us to reason about predicates and relations for arbitrary coalgebras in **Set**.

► **Example 2 (Predicates).** The fibration $\text{Pred} \rightarrow \mathbf{Set}$ of predicates has as objects in its total category Pred predicates $(P \subseteq X)$ over a set X . Each fibre Pred_X has a final object $\mathbf{1}_X = (X \subseteq X)$ and the fibred binary products are given by intersection. We note that fibred constructions, like the above products, are preserved by a change-of-base, see [41, Lem. 1.8.4]. Hence, one can also apply the results in this paper to, for example, the fibration of (binary) relations $\text{Rel} \rightarrow \mathbf{Set}$, which is given by pulling $\text{Pred} \rightarrow \mathbf{Set}$ back along the diagonal functor $\delta : \mathbf{Set} \rightarrow \mathbf{Set}$ with $\delta(I) = I \times I$. ◻

Often, one is not just interested in merely logical predicates, but rather wants to analyse quantitative aspects of system. Such predicates will be the foundation for the probabilistic μ -calculus. The following example extends the predicate fibration from Example 2 to quantitative predicates, which will give us a convenient setting to reason about quantitative properties.

► **Example 3.** We define the category of quantitative predicates qPred as follows.

$$\text{qPred} = \begin{cases} \text{objects:} & \text{pairs } (X, \delta) \text{ with } X \in \mathbf{Set} \text{ and } \delta: X \rightarrow [0, 1] \\ \text{morphisms:} & f: (X, \delta) \rightarrow (Y, \gamma) \text{ if } f: X \rightarrow Y \text{ in } \mathbf{Set} \text{ and } \delta \leq \gamma \circ f \end{cases}$$

It is easy to show that the first projection $\text{qPred} \rightarrow \mathbf{Set}$ gives rise to a cloven fibration, for which the reindexing functors are given for $u: X \rightarrow Y$ by $u^*(Y, \gamma) = (X, \lambda x. \gamma(u(x)))$. For brevity, let us refer to an object (X, δ) in qPred_X just by its underlying valuation δ . One readily checks that in qPred fibred products can be defined by $(\delta \times \gamma)(x) = \min\{\delta(x), \gamma(x)\}$ and coproducts as maximum. Fibred final objects are given by the constantly 1 valuation. \lrcorner

The original motivation for the work presented in this paper was to abstract away from the details that are involved in constructing a syntactic logic for a certain coinductive relation in [9]. In [9], the author developed a first-order logic that features the later modality to reason about program equivalences. This logic was given in a very pedestrian way, since the syntax, proof system and models were constructed from scratch. The proofs often involved phrases along the lines of “true because this is an index-wise interpretation of intuitionistic logic”. In the following example, we show how a first-order logic can be presented as a fibration, which allows us to apply the framework to a syntactic logic.

► **Example 4 (Syntactic Logic).** Suppose we are given a typed calculus, for example the simply typed λ -calculus or even the category \mathbf{Set} of sets, and a first-order logic, in which the variables range over the types of the calculus. More precisely, let Γ be a context with $\Gamma = x_1 : A_1, \dots, x_n : A_n$, where the x_i are variables and the A_i are types of the calculus. We write then $\Gamma \Vdash t : A$ if t is a term of type A in context Γ , $\Gamma \Vdash \varphi$ if φ is formula with variables in Γ , and $\Gamma \vdash \varphi$ if φ is provable in the given logic. This allows us to form a fibration as follows. First, we define \mathcal{C} to be the *syntactic category* that has context Γ as objects and tuples t of terms as morphisms $\Delta \rightarrow \Gamma$ with $\Delta \Vdash t_i : A_i$. Next, we let \mathcal{L} be the category that has pairs (Γ, φ) with $\Gamma \Vdash \varphi$ as objects, and a morphism $(\Delta, \psi) \rightarrow (\Gamma, \varphi)$ in \mathcal{L} is given by a morphism $t: \Delta \rightarrow \Gamma$ in \mathcal{C} if $\Delta \vdash \psi \rightarrow \varphi[t]$, where $\varphi[t]$ denotes the substitution of t in the formula φ . The functor $p: \mathcal{L} \rightarrow \mathcal{C}$ that maps (Γ, φ) to Γ is then easily seen to be a cloven fibration, see for example [41]. Let us assume that the logic also features a truth formula \top , conjunction \wedge and implication \rightarrow , which are subject to the usual proof rules of intuitionistic logic. We note that p has fibred finite products given by \top and conjunction. \lrcorner

As promised, the setup of Birkedal et al. [16], the topos of trees, can be recovered as an instance of our framework.

► **Example 5 (Codomain Fibration).** Let $\mathbf{Set}^{\rightarrow}$ be the arrow category over the category of sets and functions. This category has maps as objects and commuting squares as morphisms. The functor $\text{cod}: \mathbf{Set}^{\rightarrow} \rightarrow \mathbf{Set}$ that sends a map to its codomain is a fibration, in which reindexing is given by taking pullbacks, see [41]. \lrcorner

The final example will allow us to apply the framework of this paper to any category.

► **Example 6 (Trivial Fibration).** Let $\mathbf{1}$ be the final category with one object $*$ and only the identity on $*$. Then any category \mathbf{C} can be seen as fibration $!: \mathbf{C} \rightarrow \mathbf{1}$, such that fibred products etc. are normal products. \lrcorner

2.2 Coalgebras and Coinductive Predicates

Let us now introduce the second central notion of this paper: coinductive predicates. For that, we first need the notion of coalgebra [2, 42, 43, 60, 62].

► **Definition 7.** *Let $F: \mathbf{C} \rightarrow \mathbf{C}$ be a functor. A coalgebra is a morphism $c: X \rightarrow FX$. Given coalgebras $c: X \rightarrow FX$ and $d: Y \rightarrow FY$, a homomorphism from c to d is a morphism $h: X \rightarrow Y$ with $Fh \circ c = d \circ h$. We can form a category $\text{CoAlg}(F)$ of coalgebras and their homomorphisms and we call a final object in this category a final coalgebra.* \lrcorner

Coinductive predicates are easiest introduced by taking for a moment a more abstract perspective. Recall that we introduced fibrations as a way to talk abstractly about predicates, relations etc. Now we use this view to define coinductive predicates over a given coalgebra for an arbitrary notion of predicate.

► **Definition 8.** *Let $p: \mathbf{E} \rightarrow \mathbf{B}$ be a cloven fibration and $F: \mathbf{B} \rightarrow \mathbf{B}$ an endofunctor. We say that a functor $G: \mathbf{E} \rightarrow \mathbf{E}$ is a lifting of F , if $p \circ G = F \circ p$. A G -invariant in a coalgebra $c: X \rightarrow FX$ in \mathbf{B} is a $(c^* \circ G)$ -coalgebra in \mathbf{E}_X . Further, a G -coinductive predicate in c is a final $(c^* \circ G)$ -coalgebra. We often denote the carrier of the G -coinductive predicate in c by $\nu(c^* \circ G)$, see [35]. A compatible up-to technique for $c^* \circ G$ is a functor $T: \mathbf{E} \rightarrow \mathbf{E}$ with a natural transformation $T \circ c^* \circ G \Rightarrow c^* \circ G \circ T$, see [19, 59].* \lrcorner

Let us illustrate the notion of coinductive predicate in an example.

► **Example 9.** In this example, we show how the semantics of the modalities of the probabilistic modal μ -calculus ($\text{pL}\mu$) can be modelled as liftings. Given a set X , we say that a function $\rho: X \rightarrow [0, 1]$ to the unit interval is a (finitely supported) probability distributions on X , if the support $\text{supp } \rho = \{x \mid \rho(x) \neq 0\}$ is finite and $\sum_{x \in \text{supp}(\rho)} \rho(x) = 1$. One can then define a functor $\mathcal{D}: \mathbf{Set} \rightarrow \mathbf{Set}$ that maps a set to the set of all probability distributions on X . An (unlabelled) Segala system [69] or probabilistic transition system (PTS) is a coalgebra for the functor \mathcal{S} given by $\mathcal{S} = \mathcal{P} \circ \mathcal{D}$, in which states have non-deterministic transitions into probability distributions. We can now give liftings \mathcal{S}^\square and \mathcal{S}^\diamond of \mathcal{S} to qPred , which correspond to the box and diamond modality, respectively, of $\text{pL}\mu$:

$$\begin{aligned} \mathcal{S}^\square(\delta: X \rightarrow [0, 1])(D \in \mathcal{S}(X)) &= \bigwedge_{d \in D} \sum_{x \in \text{supp } d} \delta(x) \cdot d(x) \\ \mathcal{S}^\diamond(\delta: X \rightarrow [0, 1])(D \in \mathcal{S}(X)) &= \bigvee_{d \in D} \sum_{x \in \text{supp } d} \delta(x) \cdot d(x) \end{aligned}$$

Suppose now that we have a PTS $c: X \rightarrow \mathcal{S}(X)$ at hand, then $c^* \circ \mathcal{S}^\square: \text{qPred}_X \rightarrow \text{qPred}_X$ yields the expected semantics of the box modality [49]. \lrcorner

2.3 Well-Founded Induction

The final basic ingredient of this paper is well-founded induction. We will use a rather general form that is based on classes, rather than sets.

► **Definition 10.** *Let A be a class and $<$ a binary relation on A . We say that the relation $<$ is well-founded, if the well-founded induction principles holds for all $P \subseteq A$: If for all $\alpha \in A$ we have that $(\forall \beta < \alpha. \beta \in P) \implies \alpha \in P$, then $\forall \alpha \in A. \alpha \in P$.*

Given a well-founded order, we can form as usual a category from the induced partial order \leq with $\alpha \leq \beta$ if $\alpha < \beta$ or $\alpha = \beta$. Typical examples, to which the presented framework

applies, are the set ω of finite ordinals with the successor relation; the set of ordinals below any limit ordinal with their usual order; and the class of all ordinals Ord .

Recall that ordinals can be constructed as zero, successor and limit ordinals. We say that \mathbf{I} is a *classical ordinal category*¹, if every $\alpha \in \mathbf{I}$ is either zero, a successor or a limit.

3 Descending Chains in Fibrations

It is well-known that a final coalgebra of a functor F , hence also coinductive predicates, can be constructed as limits of α^{op} -chains for some limit ordinal α if such limits exist and are preserved by F [2, 3, 7]. This observation is essential to the proof approach given in this paper, as we rely on the fact that maps into a coinductive predicate, thus proofs, can alternatively be given as maps into this α^{op} -chain. In the following, we introduce the necessary machinery to leverage this fact.

More specifically, we build from a given fibration a new fibration of descending chains. In this fibration, we will be able to construct the final chain as a fixed point of a certain functor on descending chains, see Section 5.1. It should be noted though that the fibration of descending chains allows the construction of fixed points of many more functor, so called locally contractive functors. Thus, the reasoning power of the built fibration extends beyond coinductive predicates as fixed points of, e.g., contravariant functors do also exist, cf. [16, 15].

The fibration of descending chains will then admit recursive proofs for coinductive predicates and will also feature all propositional connectives and quantifiers that are present in the fibration that we started with, see Section 6. This allows us to extend any (higher-order) logic with recursive proofs for coinductive predicates.

3.1 Categories of Diagrams

Before we analyse the final chain of a functor, we introduce general diagrams and establish properties of these. We fix an index category \mathbf{I} and let $[\mathbf{I}, \mathbf{C}]$ for a category \mathbf{C} be the category of functors from \mathbf{I} to \mathbf{C} , also called the category of *\mathbf{I} -indexed diagrams in \mathbf{C}* . Given a functor $F: \mathbf{C} \rightarrow \mathbf{D}$, we define a functor $[\mathbf{I}, F]: [\mathbf{I}, \mathbf{C}] \rightarrow [\mathbf{I}, \mathbf{D}]$ on categories of diagrams by $[\mathbf{I}, F](\sigma) = F \circ \sigma$. Since $[\mathbf{I}, -]$ preserves composition of functors and applies to natural transformations, we obtain a strict 2-functor $[\mathbf{I}, -]: \mathbf{Cat} \rightarrow \mathbf{Cat}$. We use this to define for a morphism $f: X \rightarrow Y$ in \mathbf{C} , a morphism $[\mathbf{I}, f]: K_X \Rightarrow K_Y$ in $[\mathbf{I}, \mathbf{C}]$ where K_X is the constant functor sending any object in \mathbf{I} to X : Note that there is a natural transformation $K_f: K_X \Rightarrow K_Y$, which is given by $K_{f,i} = f$. Thus, we can put $[\mathbf{I}, f] = [\mathbf{I}, K_f]$.

The assignment of diagrams and lifting functors not only preserves 2-structure, but also fibrational structure.

► **Lemma 11.** *The functor $[\mathbf{I}, -]$ extends to an endomap of the fibration $\mathbf{Fib} \rightarrow \mathbf{Cat}$.*

Also adjunctions are preserved in the transition to diagrams.

► **Lemma 12.** *If $F: \mathbf{C} \rightarrow \mathbf{D}$ and $G: \mathbf{D} \rightarrow \mathbf{C}$ with $F \dashv G$, then $[\mathbf{I}, F] \dashv [\mathbf{I}, G]$.*

3.2 Descending Chains and the Later Modality

In this section, we extend the development in [16] to fibrations. We will give some intuition for the later modality and prove some basic results.

¹ We use the term “classical” here because in classical set theory, as opposed to constructive set theory, every ordinal is given in this way.

► **Assumption 13.** In the remainder of the paper, we assume that the category \mathbf{I} is induced by a well-founded class I .

In the construction of final coalgebras, one considers \mathbf{I}^{op} -indexed diagrams, which give rise to a functor $\mathbf{Cat} \rightarrow \mathbf{Cat}$ with

$$\overleftarrow{(-)} = [\mathbf{I}^{\text{op}}, -], \quad (1)$$

as in the previous section. The *category of descending chains in \mathbf{C}* is then the category $\overleftarrow{\mathbf{C}}$, the objects of which we denote by σ, τ, \dots . More explicitly, $\sigma \in \overleftarrow{\mathbf{C}}$ assigns as a functor $\sigma: \mathbf{I}^{\text{op}} \rightarrow \mathbf{C}$ to each $\alpha \in I$ an object $\sigma_\alpha \in \mathbf{C}$, and to each pair α and β with $\beta \leq \alpha$ a morphism $\sigma(\beta \leq \alpha): \sigma_\alpha \rightarrow \sigma_\beta$ in \mathbf{C} . A morphism in $\overleftarrow{\mathbf{C}}$ is a natural transformation $f: \sigma \Rightarrow \tau$, in other words, a family of morphisms $f_\alpha: \sigma_\alpha \rightarrow \tau_\alpha$ with $\tau(\beta \leq \alpha) \circ f_\alpha = f_\beta \circ \sigma(\beta \leq \alpha)$.

From Lemma 11 we get that the functor $\overleftarrow{p}: \overleftarrow{\mathbf{E}} \rightarrow \overleftarrow{\mathbf{B}}$, given by post-composition, is a fibration. The reindexing functors in this fibration will be denoted by $u^\#$. Since (co)limits are constructed point-wise in functor categories, the fibration \overleftarrow{p} inherits (co)limits from p . We obtain another fibration by change-of-base along the constant chain functor $K^{\mathbf{B}}: \mathbf{B} \rightarrow \overleftarrow{\mathbf{B}}$ that sends an object $I \in \mathbf{B}$ to the constant chain $K_I^{\mathbf{B}}: \mathbf{I}^{\text{op}} \rightarrow \mathbf{B}$ as in the diagram on the right in Figure 1.

We note the following result, which allows us to apply, for example, Lemma 12 and Theorem 27 to functors between fibres.

► **Lemma 14.** *We have that $\overleftarrow{\mathbf{E}}_I \cong \overleftarrow{\mathbf{E}}_I \cong \overleftarrow{\mathbf{E}}_{K^{\mathbf{B}}(I)}$.*

Many constructions in this paper require only limits over a bounded part of \mathbf{I}^{op} .

► **Definition 15.** *Let \mathbf{J} be a category and denote for $i \in \mathbf{J}$ by $i \downarrow \mathbf{J}$ the coslice category under i . We say that \mathbf{C} has bounded \mathbf{J} -limits, if for every $i \in \mathbf{J}$ all $(i \downarrow \mathbf{J})$ -limits exist in \mathbf{C} .*

As an example, we have for $\mathbf{I} = \omega$ and $n \in \mathbb{N}$ that $n \downarrow \omega^{\text{op}} = (\omega/n)^{\text{op}} = \underline{n}^{\text{op}}$, where \underline{n} is the set of all $k \leq n$. Hence, $n \downarrow \omega^{\text{op}}$ is finite and bounded ω^{op} -limits are finite limits.

With this definition, we can now introduce the later modality, which is the central construction that underlies the recursive proofs that we develop in this paper.

► **Theorem 16.** *Suppose that p has fibred bounded \mathbf{I}^{op} -limits. There are functors $\triangleright: \overleftarrow{\mathbf{B}} \rightarrow \overleftarrow{\mathbf{B}}$ and $\blacktriangleright: \overleftarrow{\mathbf{E}} \rightarrow \overleftarrow{\mathbf{E}}$ given on objects by*

$$(\triangleright c)_\alpha = \lim_{\beta < \alpha} c_\beta \quad \text{and} \quad (\blacktriangleright \sigma)_\alpha = \lim_{\beta < \alpha} \sigma_\beta,$$

together with natural transformations $\text{next}^\triangleright: \text{Id} \Rightarrow \triangleright$ and $\text{next}: \text{Id} \Rightarrow \blacktriangleright$. The pair $(\triangleright, \blacktriangleright)$ forms a map of fibrations $\overleftarrow{p} \rightarrow \overleftarrow{p}$ and we have $\overleftarrow{p}(\text{next}) = \text{next}^\triangleright$. Moreover, \blacktriangleright preserves fibred finite limits. Finally, if \mathbf{I} is a classical ordinal category, then \blacktriangleright has a left-adjoint \blacktriangleleft .

We note that because $\blacktriangleright: \overleftarrow{\mathbf{E}} \rightarrow \overleftarrow{\mathbf{E}}$ maps $\sigma \in \overleftarrow{\mathbf{E}}_c$ to $\blacktriangleright \sigma \in \overleftarrow{\mathbf{E}}_{\triangleright c}$, we can define a restricted version $\blacktriangleright^c: \overleftarrow{\mathbf{E}}_c \rightarrow \overleftarrow{\mathbf{E}}_c$ of the later modality that leaves the index chain untouched by putting

$$\blacktriangleright^c = (\text{next}_c^\triangleright)^\# \circ \blacktriangleright.$$

Moreover, there is a vertical natural transformation $\text{next}^c: \text{Id} \Rightarrow \blacktriangleright^c$, and \blacktriangleright^c has a left-adjoint if \mathbf{I} is classical and if p is a bifibration.

Another special case is obtained for the chains with constant index.

► **Theorem 17.** *The later modality is a strong fibred functor $\blacktriangleright: \overleftarrow{p} \rightarrow \overleftarrow{p}$ with a vertical natural transformation $\text{next}: \text{Id} \Rightarrow \blacktriangleright$, that is, $\overleftarrow{p}(\text{next}) = \text{id}$.*

Since the intention is to use Theorem 17 to extend a logic, let us present the results as proof rules. The first rule is given by the strength of \blacktriangleright , the second rule is given by composition with next, and the last rule for product preservation comes from the isomorphism in Theorem 16. This last rule can be applied in both directions, hence the indicated by double lines.

$$\frac{}{\text{mon}_{\sigma, \tau}: \sigma^\tau \rightarrow \blacktriangleright \sigma^\blacktriangleright \tau} \quad \frac{f: \tau \rightarrow \sigma}{\text{next}_\sigma \circ f: \tau \rightarrow \blacktriangleright \sigma} \quad \frac{f: \tau \rightarrow (\blacktriangleright \sigma) \times (\blacktriangleright \sigma')}{\check{f}: \tau \rightarrow \blacktriangleright (\sigma \times \sigma')}$$

The following assumption ensures that the above proof rules are available throughout the remainder of this paper.

► **Assumption 18.** p is cloven with fibred finite limits and fibred bounded \mathbf{I}^{op} -limits.

So far, we have established the fibrations and the later modality in the overview diagram in Figure 1. What remains are the adjunctions that relate the fibrations, cf. [41, Exercise 1.8.8].

► **Theorem 19.** *If \mathbf{E} has fibred \mathbf{I}^{op} -limits, then $K^{\mathbf{E}}: \mathbf{E} \rightarrow \overline{\mathbf{E}}$ has a fibred right adjoint $L^{\mathbf{E}}$, given by $L^{\mathbf{E}}(\sigma) = \lim_{\alpha \in \mathbf{I}} \sigma_\alpha$. If \mathbf{B} has \mathbf{I}^{op} -limits, then $K^{\mathbf{B}}: \mathbf{B} \rightarrow \overline{\mathbf{B}}$ and $I: \overline{\mathbf{E}} \rightarrow \overline{\mathbf{B}}$ have right adjoints $L^{\mathbf{B}}$ and R , given by $L^{\mathbf{B}}(c) = \lim_{\alpha \in \mathbf{I}} c_\alpha$ and $R = \pi^\#$, where $\pi_\beta: \lim_{\alpha \in \mathbf{I}} c_\alpha \rightarrow c_\beta$ are the limit projections and $(-)^{\#}$ is reindexing in \overleftarrow{p} .*

4 Cartesian Closure and the Löb Rule

Up to this point, we have only shown the existence of the next and monotonicity rule that we used in the example in the introduction. What is missing is the recursion given in form of the Löb rule. The goal of this section is to establish the recursion mechanism by utilising so-called Löb induction, which is based on the later modality that we introduced in Section 3.2. To state and prove the Löb induction, we need exponential objects in our fibration $\overleftarrow{p}: \overline{\mathbf{E}} \rightarrow \overline{\mathbf{B}}$ of chains. In the first part of this section, we show how to construct these from exponential objects in $p: \mathbf{E} \rightarrow \mathbf{B}$. The second part is devoted to establishing the Löb rule.

4.1 Fibred Cartesian Closure of Diagrams

A *fibred Cartesian closed category (fibred CCC)* is a fibration $p: \mathbf{E} \rightarrow \mathbf{B}$ in which every fibre is Cartesian closed and reindexing preserves this structure, see [41, Def. 1.8.2]. In a fibred CCC we can model in particular implication, which is what we will need to formulate the Löb rule below. Given a fibred CCC, we show now that the fibration of diagrams is also a fibred CCC. Since the construction of exponential objects in categories of diagrams does not depend on working with a well-founded index category, we will formulate the results in this section for an arbitrary index category \mathbf{I} , like we did in Section 3.1.

Let $S: \mathbf{I}^{\text{op}} \times \mathbf{I} \rightarrow \mathbf{C}$ be a functor. The *end* of S is an object $\int_{i \in \mathbf{I}} S(i, i)$ in \mathbf{C} together with a universal extranatural transformation $\pi: \int_{i \in \mathbf{I}} S(i, i) \rightarrow S$. This means that π is a family of morphisms indexed by objects in \mathbf{I} , such that the diagram below commutes for all $u: i \rightarrow j$.

$$\begin{array}{ccc} \int_{i \in \mathbf{I}} S(i, i) & \xrightarrow{\pi_j} & S(j, j) \\ \downarrow \pi_i & & \downarrow S(u, \text{id}) \\ S(i, i) & \xrightarrow{S(\text{id}, u)} & S(i, j) \end{array}$$

Given another extranatural transformation $\alpha: X \rightarrow S$ there is a unique $f: X \rightarrow \int_{i \in \mathbf{I}} S(i, i)$ with $\pi_i \circ f = \alpha_i$ for every $i \in \mathbf{I}$. It is well-known that ends can be computed as certain limits in \mathbf{C} . By analysing carefully the necessary limits, we obtain the following result.

► **Theorem 20.** Let \mathbf{I} be a category and $p: \mathbf{E} \rightarrow \mathbf{B}$ a cloven fibration that has fibred equalisers, fibred exponents and fibred bounded \mathbf{I} -products. Then $[\mathbf{I}, p]: [\mathbf{I}, \mathbf{E}] \rightarrow [\mathbf{I}, \mathbf{B}]$ is again a fibred CCC. The exponential object of $F, G \in [\mathbf{I}, \mathbf{E}]_U$ is given by

$$(G^F)(i) = \int_{v: i \rightarrow j} (U(v)^* G(j))^{U(v)^* F(j)}.$$

► **Assumption 21.** In the remainder we additionally assume that $p: \mathbf{E} \rightarrow \mathbf{B}$ is a fibred CCC, has fibred equalisers and fibred bounded \mathbf{I} -products.

From Assumption 21, we get that \overleftarrow{p} is a fibred CCC. Since change-of-base also preserves fibred exponentials, the fibration \overleftarrow{p} that we obtained by pulling \overleftarrow{p} back along the constant chain functor in Section 3.2 is also a fibred CCC, see [41, Lem. 1.8.4] and [71].

► **Example 22.** Fibred exponentials exist in Pred_X with $Q^P = \{x \in X \mid x \in P \implies x \in Q\}$. The fibration $\overleftarrow{\text{Pred}}$ consists then of descending chains of predicates. This means that if $\sigma \in \overleftarrow{\text{Pred}}_X$, then σ is a chain with $\sigma_0 \supseteq \sigma_1 \supseteq \dots$. Note now that each fibre Pred_X is a complete lattice, hence equalisers are trivial and (bounded) limits are just given as (bounded) infima. Hence, Theorem 20 applies and we obtain that $\overleftarrow{\text{Pred}}$ is a fibred CCC. Since equalisers are trivial, it is easy to see that the exponential for $\sigma, \tau \in \overleftarrow{\text{Pred}}_X$ can be defined as follows.

$$(\tau^\sigma)_n = \bigcap_{m \leq n} \tau_m^{\sigma_m}$$

Since fibred exponentials are preserved by a change-of-base, see [41, Lem. 1.8.4], they also exist in the fibration of relations $\text{Rel} \rightarrow \mathbf{Set}$ and the associated fibration $\overleftarrow{\text{Rel}} \rightarrow \overleftarrow{\mathbf{Set}}$. ◻

► **Example 23.** Recall that we defined in Example 3 a category of quantitative predicates. We note that this fibration is a fibred CCC with exponents given by

$$(\delta \Rightarrow \gamma)(x) = \begin{cases} 1, & \delta(x) \leq \gamma(x) \\ \gamma(x), & \text{otherwise} \end{cases}.$$

Again, each fibre qPred_X is a complete lattice and so $\overleftarrow{\text{qPred}}$ is a fibred CCC for any \mathbf{I} . ◻

► **Example 24.** In Example 4, we defined a fibration $p: \mathcal{L} \rightarrow \mathcal{C}$ for a first-order logic with conjunction and implication. From the implication we obtain that p is a fibred CCC. Moreover, since each fibre is a pre-order, equalisers are again trivial. If \mathbf{I} is the poset ω of finite ordinals, then \overleftarrow{p} is a fibred CCC. Explicitly, for chains φ, ψ of formulas in \overleftarrow{p}_A above a type A , the exponent $\psi \Rightarrow \varphi$ in \overleftarrow{p} is given by a finite conjunction:

$$(\psi \Rightarrow \varphi)_n = \bigwedge_{m \leq n} \psi_m \rightarrow \varphi_m. \quad \text{◻}$$

► **Example 25.** The trivial fibration is a fibred CCC if and only if \mathbf{C} is Cartesian closed. In this case, the end formula reduces to $(G^F)(i) = \int_{v: j \rightarrow i} G(j)^{F(j)}$ for $G, F: \mathbf{I}^{\text{op}} \rightarrow \mathbf{C}$. ◻

4.2 The Löb Rule

One purpose of the later modality is that it allows us to characterise maps in \overleftarrow{p} , so-called contractive maps, of which we can construct fixed points.

► **Definition 26.** A map $f: \tau \times \sigma \rightarrow \sigma$ in $\overleftarrow{\mathbf{E}}_c$ is called g -contractive if $g: \tau \times \blacktriangleright^c \sigma \rightarrow \sigma$ with $f = g \circ (\text{id} \times \text{next}_\sigma)$. We call $s: \tau \rightarrow \sigma$ a fixed point or solution for f , if $s = f \circ \langle \text{id}, s \rangle$. ◻

We can now show that there is a operator in \overleftarrow{p} that allows us to construct fixed points.

► **Theorem 27.** *For every $\sigma \in \overleftarrow{\mathbf{E}}_c$ there is a unique morphism $\text{löb}_\sigma^c: \sigma \blacktriangleright^c \sigma \rightarrow \sigma$ in $\overleftarrow{\mathbf{E}}_c$, such that for all g -contractive maps f the map $\text{löb}_\sigma^c \circ \lambda g$ is a solution for f . From this we obtain every for $\sigma \in \overleftarrow{\mathbf{E}}_X$ a unique morphism $\text{löb}_\sigma: \sigma \blacktriangleright \sigma \rightarrow \sigma$ that solves any contractive map in $\overleftarrow{\mathbf{E}}_X$.*

► **Proposition 28.** *The morphisms löb^c and löb are dinatural transformations.*

From Theorem 27, we obtain the Löb proof rule. This rule allows us to introduce recursion into proofs, by giving us the proof goal σ as an assumption guarded by the later modality.

$$\frac{f: \tau \times \blacktriangleright^c \sigma \rightarrow \sigma}{\text{löb}_\sigma^c \circ \lambda f: \tau \rightarrow \sigma} \quad \text{with} \quad \text{löb}_\sigma^c \circ \lambda f = f \circ (\text{id} \times \text{next}_\sigma) \circ \langle \text{id}, \text{löb}_\sigma^c \circ \lambda f \rangle$$

5 Locally Contractive Functors and Coinduction

One of the central notions of Birkedal et al. [16] is that of locally contractive functors. Such functors admit fixed points in the topos of trees and are closed under various constructions like composition and products. Locally contractive functors are used in [16] as a different way of solving recursive domain equations, which is where the name ‘‘synthetic domain theory’’ comes from. In this section, we restate the definition of contractive functors, and generalise the fixed point construction and the closure properties to the fibrations \overleftarrow{p} and \overline{p} .

In the following, we use the natural transformation $\text{comp}_{X,Y,Z}: X^Y \times Z^X \rightarrow Z^Y$ that composes internal morphisms. We will refer to the isomorphism $\blacktriangleright \sigma \times \blacktriangleright \tau \rightarrow \blacktriangleright (\sigma \times \tau)$ as $\delta^\blacktriangleright$.

► **Definition 29.** *A functor $F: \overleftarrow{\mathbf{C}} \rightarrow \overleftarrow{\mathbf{C}}$ is called locally contractive if F is strong, there is a natural transformation $C_{\sigma,\tau}^F: \blacktriangleright (\sigma^\tau) \rightarrow F \sigma^{F\tau}$ with $\text{st}_{\sigma,\tau}^F = C_{\sigma,\tau}^F \circ \text{next}_{\sigma\tau}$, and fulfils $C_{\sigma,\sigma}^F \circ \blacktriangleright \ulcorner \text{id} \urcorner = \ulcorner \text{id} \urcorner$ and $\text{comp} \circ (C_{\sigma,\tau}^F \times C_{\gamma,\sigma}^F) = C_{\gamma,\tau}^F \circ \blacktriangleright \text{comp} \circ \delta^\blacktriangleright$. A lifting $(F, G): \overleftarrow{p} \rightarrow \overleftarrow{p}$ is locally contractive if (F, G) is strong, F and G are locally contractive and $\overleftarrow{p} C^G = C^F$.*

The next theorem records the essential closure properties of locally contractive functors.

► **Theorem 30.** *Let $F, G: \mathbf{C} \rightarrow \mathbf{C}$ be functors. If F or G is locally contractive, then $F \circ G$ is; if F and G are locally contractive, then $F \times G$ is. Both, $(\blacktriangleright, \blacktriangleright): \overleftarrow{p} \rightarrow \overleftarrow{p}$ and $\blacktriangleright: \overline{p} \rightarrow \overline{p}$ are locally contractive. The constant functor $\lambda \tau. \sigma$ is locally contractive for any $\sigma \in \overleftarrow{\mathbf{E}}$.*

The proof of the following theorem proceeds in the same way as the one given in [16] by first establishing for all $\alpha \in \mathbf{I}$ and $\beta < \alpha$ that a locally contractive functor G maps any β -isomorphism f to an α -isomorphism Gf above the corresponding α -iso $F(\overleftarrow{p} f)$. An α -isomorphism is thereby a morphism $f: \sigma \rightarrow \tau$, such that for all $\beta \leq \alpha$ all f_β are isomorphisms.

► **Theorem 31.** *Any locally contractive lifting (F, G) has a unique fixed point in $\overleftarrow{\mathbf{E}}$.*

In Section 7, we will need the following version on fibres for the semantics of $\text{pL}\mu$.

► **Theorem 32.** *For any $c \in \overleftarrow{\mathbf{B}}$ and locally contractive functor $F: \overleftarrow{\mathbf{E}}_c \rightarrow \overleftarrow{\mathbf{E}}_c$ a unique fixed point of F exists in $\overleftarrow{\mathbf{E}}_c$. Consequently, also locally contractive functors on $\overleftarrow{\mathbf{E}}_X$ for $X \in \mathbf{B}$ have unique fixed points by using that $\overleftarrow{\mathbf{E}}_X \cong \overleftarrow{\mathbf{E}}_{K\mathbf{B}(X)}$.*

5.1 The Final Chain and Up-To Techniques

Having laid the ground work, we come to the objects of interest: coinductive predicates. The following definition captures the usual construction of the final chain. Recall that $\overleftarrow{(-)}$ is a functor $\mathbf{Fib} \rightarrow \mathbf{Fib}$. Thus, from $\Phi: \mathbf{E}_I \rightarrow \mathbf{E}_I$, we obtain $\overleftarrow{\Phi}: \overleftarrow{\mathbf{E}}_I \rightarrow \overleftarrow{\mathbf{E}}_I$ by Lemma 14. The functor $\overleftarrow{\Phi}$ applies thereby Φ point-wise to chains.

► **Definition 33.** Given a functor $\Phi: \mathbf{E}_I \rightarrow \mathbf{E}_I$, we define the final chain of Φ to be the fixed point $\nu(\blacktriangleright \overleftarrow{\Phi})$ of the locally contractive functor $\blacktriangleright \circ \overleftarrow{\Phi}$.

We can now construct an adjunction between Φ -invariants and coalgebras for $\blacktriangleright \overleftarrow{\Phi}$, cf. [44]. This is a slightly more expressive version of the usual construction of final coalgebras.

► **Theorem 34.** Suppose $\Phi: \mathbf{E}_I \rightarrow \mathbf{E}_I$ preserves \mathbf{I}^{op} -limits. Then the adjunction $K^{\mathbf{E}} \dashv L^{\mathbf{E}}$ lifts to an adjunction $\hat{K}^{\mathbf{E}} \dashv \hat{L}^{\mathbf{E}}$ between the categories $\text{CoAlg}(\Phi)$ and $\text{CoAlg}(\blacktriangleright \overleftarrow{\Phi})$ of Φ - and $\blacktriangleright \overleftarrow{\Phi}$ -coalgebras. This gives $\nu\Phi \cong \hat{L}^{\mathbf{E}}(\nu(\blacktriangleright \overleftarrow{\Phi}))$, where $\nu(\blacktriangleright \overleftarrow{\Phi})$ is the unique fixed point of $\blacktriangleright \overleftarrow{\Phi}$.

Theorem 34 will play a central role in recursive proofs, as it allows us to express maps into $\nu\Phi$ in terms of maps into $\nu(\blacktriangleright \overleftarrow{\Phi})$, and it allows us to unfold the final chain and thereby to make progress in a proof. Just as important as unfolding is the ability to reason inside syntactic contexts, use transitivity of relations etc. in a proof. Such properties are captured by up-to techniques, see Definition 8.

► **Theorem 35.** Let T and Φ be functors $\mathbf{E}_I \rightarrow \mathbf{E}_I$. If there is a natural transformation $\rho: T\Phi \Rightarrow \Phi T$, then there is a map $\hat{\rho}: \overleftarrow{T}\nu(\blacktriangleright \overleftarrow{\Phi}) \rightarrow \nu(\blacktriangleright \overleftarrow{\Phi})$ in $\overline{\mathbf{E}}_I$.

► **Remark 36.** Pous and Rot [56] prove a result similar to Theorem 35, namely that a monotone function T on a complete lattice is below the companion of Φ if and only if there is a map $\overleftarrow{T}\nu(\blacktriangleright \overleftarrow{\Phi}) \rightarrow \nu(\blacktriangleright \overleftarrow{\Phi})$. This is equivalent to Theorem 35 because the companion is compatible. \lrcorner

From Theorems 19, 34 and 35 we obtain the following proof rules, where the first initialises a proof by moving from a coinductive predicate to the final chain.

$$\frac{K_A \longrightarrow \nu(\blacktriangleright \overleftarrow{\Phi})}{A \longrightarrow \nu\Phi} \quad \frac{f: \tau \rightarrow \blacktriangleright \overleftarrow{\Phi}(\nu(\blacktriangleright \overleftarrow{\Phi}))}{f: \tau \rightarrow \nu\blacktriangleright \overleftarrow{\Phi}} \quad \frac{\rho: T\Phi \Rightarrow \Phi T \quad f: \tau \rightarrow \overleftarrow{T}\nu(\blacktriangleright \overleftarrow{\Phi})}{\overleftarrow{\rho} \circ f: \tau \rightarrow \nu(\blacktriangleright \overleftarrow{\Phi})}$$

The last result in this section, recorded here for completeness, allows us to obtain compatible up-to techniques on fibres from global up-to techniques.

► **Proposition 37.** Let $(F, G): p \rightarrow p$ be a map of fibrations, $c: I \rightarrow FI$ a coalgebra in \mathbf{B} , and $T: \mathbf{E} \rightarrow \mathbf{E}$ a lifting of the identity $\text{Id}_{\mathbf{E}}$. Define $\Phi := c^* \circ G: \mathbf{E}_I \rightarrow \mathbf{E}_I$ to be the predicate transformer associated to c , see Definition 8. If there is a vertical natural transformation $\rho: TG \Rightarrow GT$, then there is a vertical natural transformation $\rho^c: T\Phi \Rightarrow \Phi T$.

6 Chains in First-Order Fibrations

The goal of this section is to show that the fibration $\bar{p}: \overline{\mathbf{E}} \rightarrow \mathbf{B}$ of \mathbf{I}^{op} -chains with constant index is a first-order fibration (FO fibration) if $p: \mathbf{E} \rightarrow \mathbf{B}$ is an FO fibration. This allows us to construct out of a given FO logic another FO logic that features the later modality.

6.1 Products, Coproducts and Quantifiers for Descending Chains

Because of Lemma 14, we can apply many construction easily point-wise to chains with constant index. For instance, we can lift products and coproducts in the following sense.

► **Theorem 38.** If for $u: I \rightarrow J$ in \mathbf{B} the coproduct $\coprod_u: \mathbf{E}_I \rightarrow \mathbf{E}_J$ along u exists, then the coproduct $\coprod_u: \overline{\mathbf{E}}_I \rightarrow \overline{\mathbf{E}}_J$ along u is given by $\overleftarrow{\coprod}_u$. Similarly, the product \prod_u along u is $\overleftarrow{\prod}_u$.

► **Example 39.** Both Pred and qPred to have products and coproducts along any function in \mathbf{Set} . For instance, products in qPred along functions $u: X \rightarrow Y$ are given by

$$\prod_u (\delta: X \rightarrow [0, 1])(y) = \inf\{\delta(x) \mid x \in X, u(x) = y\}.$$

In a syntactic logic, Example 4, one has that $\mathcal{L} \rightarrow \mathcal{C}$ products and coproducts along projections $(\Gamma, x : A) \rightarrow \Gamma$ are universal and existential quantification over A , respectively. Arbitrary (co)products can then be defined in terms of the equality relation in the logic, cf. [41]. By Theorem 38, all these products and coproducts lift to the fibrations of descending chains. \square

Let us denote for $I \in \mathbf{B}$ the later modality on $\bar{\mathbf{E}}_I$ by \blacktriangleright^I . We can then establish the following essential properties about the interaction of the later modalities and (co)products, which are analogue to those in [16, Thm. 2.7]. This theorem allows one to distribute in proofs quantifiers over the later modality.

► **Theorem 40.** *The following holds for fibred products and coproducts in \bar{p} .*

- *There is an isomorphism $\blacktriangleright^J \circ \prod_u \cong \prod_u \circ \blacktriangleright^I$.*
- *There is a natural transformation $\iota: \prod_u \circ \blacktriangleright^I \Rightarrow \blacktriangleright^J \circ \prod_u$. Moreover, if u is inhabited, that is, has a section $v: J \rightarrow I$, then ι has a section ι^v .*

For $u: I \rightarrow J$ in \mathbf{B} , we can present the central results of this section as proof rules:

$$\frac{f: \tau \rightarrow u^* \sigma}{\check{f}: \prod_u \tau \rightarrow \sigma} \quad \frac{f: \tau \rightarrow \prod_u (\blacktriangleright^I \sigma)}{\iota \circ f: \tau \rightarrow \blacktriangleright^J (\prod_u \sigma)} \quad \frac{f: u^* \tau \rightarrow \sigma}{\check{f}: \tau \rightarrow \prod_u \sigma} \quad \frac{f: \tau \rightarrow \blacktriangleright^J (\prod_u \sigma)}{\check{f}: \tau \rightarrow \prod_u (\blacktriangleright^I \sigma)}$$

6.2 First Order Fibration of Descending Chains

As the name suggests, a first-order fibration models first-order logic with equality. Such an FO fibration is a fibration $p: \mathbf{E} \rightarrow \mathbf{B}$, which is a fibred pre-ordered lattice and fibred CCC, and has products and coproducts, which satisfy the Beck-Chevalley and Frobenius conditions, along all morphisms in \mathbf{B} , see [41, Def. 4.2.1] for details. We now show that not only is the fibration of constant-index chains in p an FO fibration, but is also strongly related to p .

► **Theorem 41.** *If $p: \mathbf{E} \rightarrow \mathbf{B}$ is an FO fibration, then $\bar{p}: \bar{\mathbf{E}} \rightarrow \mathbf{B}$ is as well an FO fibration. Furthermore, if the fibred coproducts and coproducts along morphisms preserve \mathbf{I}^{op} -limits, then $L^{\mathbf{E}}: \bar{\mathbf{E}} \rightarrow \mathbf{E}$ preserves all the FO structure except for implication. For implication, truth is preserved, i.e., for all $\sigma, \tau \in \bar{\mathbf{E}}_I$ there is a morphism $L(\sigma^\tau) \rightarrow L\sigma^{L\tau}$. If $\tau = K_X^{\mathbf{E}}$ for some $X \in \mathbf{E}_I$, then this morphism is an isomorphism. Finally, $K^{\mathbf{E}}$ is a fully faithful functor.*

That preservation of exponentials fails can be seen by taking $\sigma, \tau \in [\omega^{\text{op}}, \text{Pred}_{\mathbb{N}}]$ to be $\tau_n = \mathbb{N} \setminus \{1, \dots, n\}$ and $\sigma_n = \{0\}$. Then $L(\sigma^\tau) = \{0\}$ but $L\sigma^{L\tau} = \mathbb{N}$.

7 Examples

In this section, we show the framework in action. Specifically, we show how a novel proof system for the probabilistic modal μ -calculus $\text{pL}\mu$ can be obtained, and we show a language and its semantics for probabilistic productive coinductive programming.

7.1 Recursive Proofs for the Probabilistic Modal μ -Calculus

The probabilistic modal μ -calculus $\text{pL}\mu$ has exactly the same syntax as the modal μ -calculus $\text{L}\mu$. However, formulas are interpreted as probability distributions [39]. We extend the

$$\begin{array}{c}
\frac{}{\triangleright \gamma, \psi \vdash \psi} \text{ (Pr)} \\
\frac{}{\triangleright \gamma, \psi \vdash \varphi(\psi)} \text{ assumption} \\
\frac{}{\triangleright \gamma, \psi \vdash \varphi(\triangleright \psi)} \varphi \text{ positive} + \text{ (Next)} \\
\frac{}{\triangleright \gamma, \psi \vdash \varphi(\triangleright \nu X. \varphi(X))} \text{ (Pr)} \\
\frac{}{\triangleright \gamma, \psi \vdash \triangleright(\psi \rightarrow \nu X. \varphi(X))} \text{ (Mon)} \\
\frac{}{\triangleright \gamma, \psi \vdash \triangleright \psi \rightarrow \triangleright \nu X. \varphi(X)} \varphi \text{ positive} \\
\frac{}{\triangleright \gamma, \psi \vdash \varphi(\triangleright \nu X. \varphi(X))} \text{ (Step)} \\
\frac{}{\triangleright \gamma, \psi \vdash \nu X. \varphi(X)} (\rightarrow\text{-I}) \\
\frac{}{\triangleright \gamma \vdash \gamma} \text{ (Löb)} \\
\frac{}{\vdash \gamma}
\end{array}$$

■ **Figure 3** $\varphi(X)$ positive in X , ψ $L\mu$ -formula, $\gamma = \psi \rightarrow \nu X. \varphi(X)$ with assumption $\psi \rightarrow \varphi(\psi)$.

coinductive fragment of $pL\mu$ here with the later modality and thereby obtain the following formulas over sets At and Var of propositional variables P and fixed point variables X :

$$\varphi, \psi ::= P \mid \overline{P} \mid X \mid \top \mid \perp \mid \nu X. \varphi \mid \triangleright \varphi \mid \square \varphi \mid \diamond \varphi \mid \varphi \sqcap \psi \mid \varphi \sqcup \psi \mid \varphi \rightarrow \psi,$$

where X must occur positively in φ when forming $\nu X. \varphi$. Given a formula φ with no or one free variable² X , a Segala system $c: Q \rightarrow \mathcal{S}(Q)$ and an interpretation $I: Q \rightarrow \text{qPred}_{\text{At}}$, we use Theorem 30 to define a locally contractive functor $\llbracket \varphi \rrbracket: \text{Pred}_Q^n \rightarrow \text{Pred}_Q$ with $n = 0, 1$, where we only display the interesting cases. The remaining cases are given in Appendix A.

$$\llbracket P \rrbracket = K(I(P)) \quad \llbracket X \rrbracket = \triangleright \quad \llbracket \nu X. \varphi \rrbracket = \nu \llbracket \varphi \rrbracket \quad \llbracket \square \varphi \rrbracket = c^\# \circ \overleftarrow{\mathcal{S}^\square} \circ \llbracket \varphi \rrbracket$$

This definition and the previous development gives us that the following rules are sound for this interpretation, where double lines are rules that can be used in both directions.

$$\begin{array}{c}
\frac{\Delta \vdash \varphi[\triangleright \nu X. \varphi/X]}{\Delta \vdash \nu X. \varphi} \text{ (Step)} \quad \frac{\Delta \vdash \varphi}{\Delta \vdash \triangleright \varphi} \text{ (Next)} \quad \frac{\Delta \vdash \triangleright(\varphi \rightarrow \psi)}{\Delta \vdash \triangleright \varphi \rightarrow \triangleright \psi} \text{ (Mon)} \\
\frac{\Delta, \triangleright \varphi \vdash \varphi}{\Delta \vdash \varphi} \text{ (Löb)} \quad \frac{\Delta \vdash \square \triangleright \varphi}{\Delta \vdash \triangleright \square \varphi} \quad \frac{\Delta \vdash \diamond \triangleright \varphi}{\Delta \vdash \triangleright \diamond \varphi} \quad + \text{ normal, intuitionistic modal logic}
\end{array}$$

In Figure 3, we show how Park's rule can be proven from these rules. Theorem 41 gives us that these rules are sound and their semantics are complete for the standard semantics of formulas that only have constant premises, i.e. pure modal formulas, in implications.

Let us make two final remarks about this example. First, note that the implication is an internalisation of the ordering on quantitative predicates and thus has, a priori, nothing to do with probabilities. In particular, we have $\llbracket \overline{P} \rrbracket \neq \llbracket P \rightarrow \perp \rrbracket$. Second, the proof rules give rise to a constructive and recursive proof system for $pL\mu$. This is insofar interesting, as that the completeness proof for Kozen's axiomatisation for $L\mu$ is non-constructive, and a non-probabilistic version of the above presented proof system may give new insights, cf. [27]. Also an analogous version of our cut-free proof system for Horn clause theories [10] may shed new light on cut-free proofs for $(p)L\mu$, cf. [4].

7.2 Probabilistic Productive Coinductive Programming

In this last example, we show how one can obtain a new programming language for higher-order probabilistic programming with coinductive types, in which all programs are terminating.

² We restrict ourselves to this case for simplicity. Supporting several variables is a direct generalisation.

This is in contrast to the language provided in [74], where full recursion is essential to coinductive programming. Full recursion introduces, however, non-terminating and non-productive programs, which makes reasoning about programs unnecessarily difficult [73], especially in the probabilistic setting. As such, the total programming language, which we are about to introduce, provides us with coinductive, probabilistic types, while retaining the good properties of terminating and productive programs.

The essential ingredient are so-called *quasi-Borel spaces* that were introduced by Heunen et al. [37] as a setting for higher-order probabilistic programming. In particular, the category \mathbf{qBS} of quasi-Borel spaces and their morphisms is (co)complete and Cartesian closed, see [37, 74] for details. From the framework, we obtain that $\overline{\mathbf{qBS}} = [\omega^{\text{op}}, \mathbf{qBS}]$ is as well a (co)complete CCC with later modality and Löb rule. This allows us to provide a probabilistic higher-order programming language with coinductive types.

This language has types and terms that are given in Appendix B. One coinductive example given in [74] is that of a random walk, which produces a stream of random positions for a given standard deviation σ . We may define the type \mathbb{R}^ω of \mathbb{R} -valued streams as fixed point type by $\mathbb{R}^\omega = \text{fix } X. \mathbb{R} \times \blacktriangleright X$. A random walk can be produced by the following guarded recursive program $\text{RW} : \mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{R}^\omega$, where $\text{normal}\langle \rho, \sigma \rangle$ draws from a normal distribution with expected value ρ and standard deviation σ .

$$\text{RW} = \lambda \sigma. \text{fix } f : \blacktriangleright (\mathbb{R} \rightarrow \mathbb{R}^\omega). \lambda x. \text{in } \langle x, f \otimes \text{next}(\text{normal}\langle x, \sigma \rangle) \rangle$$

The details of how the above types and terms can be interpreted in $\overline{\mathbf{qBS}}$ are given in Appendix B. Since \mathbf{qBS} is complete, we thus obtain an interpretation of the types and terms in \mathbf{qBS} , which corresponds to the expected final coalgebra semantics, see Theorem 34.

8 Conclusion and Future Work

In this paper, we have established a framework that allows us to reason about coinductive predicates in many cases by using recursive proofs. At the heart of this approach sits the so-called later modality, which comes from provability logic [12, 68, 70] but was later used to obtain guarded recursion in type theories [5, 6, 17, 52] and in domain theory [15, 16]. This modality allows us to control the recursion steps in a proof without having to invoke parity or similar conditions [22, 28, 64, 67], as we have seen in the examples in Section 7. Moreover, even though Birkedal et al. [16] obtained similar results, their framework is limited to \mathbf{Set} -valued presheaves, while our results are applicable in a much wider range of situations. In particular, we were able to devise a novel probabilistic programming language that guarantees productivity on coinductive types.

So what is there left to do? For once, we have not touched upon how to automatically extract a syntactic logic and models from the fibration $\overline{\mathcal{L}} \rightarrow \overline{\mathcal{C}}$ obtained in Example 24. This would subsume and simplify much of the development in [9]. Next, we only proved the existence of quantifiers that range over fixed domains. It would be useful to extend this construction to indexed domains to, for example, obtain Kripke models abstractly. However, such a construction would be similar to that of exponents in Theorem 20 and thus quite involved. At the same time, also a category theoretical analysis of the delayed implication in [23] is needed. Also a closer analysis of the relation to proof systems obtained through parameterised coinduction, the companion or cyclic proof systems may shed some light on the strength of the proof approach presented in this paper. Such an analysis requires to understand how the causal proofs that the presented framework and the companion support [56], and parameterised coinduction are related. Finally, after a few first step into

the direction of proof search for coinductive Horn clause theories in [10], the results of the present paper need to be applied to obtain proof search procedures for other logics and theories.

References

- 1 Andreas Abel, Brigitte Pientka, David Thibodeau, and Anton Setzer. Copatterns: Programming Infinite Structures by Observations. In *POPL'13*, pages 27–38. ACM, 2013. doi:10.1145/2429069.2429075.
- 2 Jiří Adámek, Stefan Milius, and Lawrence S. Moss. Introduction to Category Theory, Algebras and Coalgebra. *A monograph in preparation*, 2010. URL: http://www.tu-braunschweig.de/Medien-DB/iti/survey_full.pdf.
- 3 Jiří Adámek and Vera Trnková. Initial Algebras and Terminal Coalgebras in Many-Sorted Sets. *MSCS*, 21(2):481–509, 2011. doi:10.1017/S0960129510000502.
- 4 Bahareh Afshari and Graham E. Leigh. Cut-Free Completeness for Modal Mu-Calculus. In *Proc. of LICS'17*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005088.
- 5 Andrew W. Appel, Paul-André Melliès, Christopher D. Richards, and Jérôme Vouillon. A Very Modal Model of a Modern, Major, General Type System. In *POPL*, pages 109–122. ACM, 2007. doi:10.1145/1190216.1190235.
- 6 Robert Atkey and Conor McBride. Productive Coprogramming with Guarded Recursion. In *ICFP*, pages 197–208. ACM, 2013. doi:10.1145/2500365.2500597.
- 7 Michael Barr. Terminal Coalgebras in Well-Founded Set Theory. *TCS*, 114(2):299–315, 1993. doi:10.1016/0304-3975(93)90076-6.
- 8 Henning Basold. Code Repository, 2018. URL: <https://perso.ens-lyon.fr/henning.basold/code/>.
- 9 Henning Basold. *Mixed Inductive-Coinductive Reasoning: Types, Programs and Logic*. PhD Thesis, Radboud University, 2018. URL: <https://hdl.handle.net/2066/190323>.
- 10 Henning Basold, Ekaterina Komendantskaya, and Yue Li. Coinduction in Uniform: Foundations for Corecursive Proof Search with Horn Clauses. In *ESOP'19*, volume 11423 of *LNCS*. Springer, 2019. arXiv:1811.07644.
- 11 Henning Basold, Damien Pous, and Jurriaan Rot. Monoidal Company for Accessible Functors. In *CALCO 2017*, volume 72 of *LIPICs*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPICs.CALCO.2017.5.
- 12 Lev D. Beklemishev. Parameter Free Induction and Provably Total Computable Functions. *TCS*, 224(1-2):13–33, 1999. doi:10.1016/S0304-3975(98)00305-3.
- 13 Jean Bénabou. Fibered Categories and the Foundations of Naive Category Theory. *Journal of Symbolic Logic*, 50(1):10–37, 1985. doi:10.2307/2273784.
- 14 Lars Birkedal, Alés Bizjak, Ranald Clouston, Hans Bugge Grathwohl, Bas Spitters, and Andrea Vezzosi. Guarded Cubical Type Theory: Path Equality for Guarded Recursion. In *CSL 2016*, volume 62 of *LIPICs*, pages 23:1–23:17. Schloss Dagstuhl, 2016. doi:10.4230/LIPICs.CSL.2016.23.
- 15 Lars Birkedal and Rasmus Ejlers Møgelberg. Intensional Type Theory with Guarded Recursive Types qua Fixed Points on Universes. In *LICS*, pages 213–222. IEEE Computer Society, 2013. doi:10.1109/LICS.2013.27.
- 16 Lars Birkedal, Rasmus Ejlers Møgelberg, Jan Schwinghammer, and Kristian Støvring. First Steps in Synthetic Guarded Domain Theory: Step-Indexing in the Topos of Trees. *LMCS*, 8(4), 2012. doi:10.2168/LMCS-8(4:1)2012.
- 17 Ales Bizjak, Hans Bugge Grathwohl, Ranald Clouston, Rasmus Ejlers Møgelberg, and Lars Birkedal. Guarded Dependent Type Theory with Coinductive Types. In *FoSSaCS*, volume 9634 of *LNCS*, pages 20–35. Springer, 2016. arXiv:1601.01586.
- 18 Jasmin Christian Blanchette, Johannes Hölzl, Andreas Lochbihler, Lorenz Panny, Andrei Popescu, and Dmitriy Traytel. Truly Modular (Co)Datatypes for Isabelle/HOL. In Gerwin

- Klein and Ruben Gamboa, editors, *Proceedings of ITP 2014*, volume 8558 of *LNCS*, pages 93–110. Springer, 2014. doi:10.1007/978-3-319-08970-6_7.
- 19 Filippo Bonchi, Daniela Petrişan, Damien Pous, and Jurriaan Rot. Coinduction Up-to in a Fibrational Setting. In *LICS '14*, pages 20:1–20:9. ACM, 2014. doi:10.1145/2603088.2603149.
 - 20 Julian C. Bradfield and Colin Stirling. Modal Mu-Calculi. In *Handbook of Modal Logic*, pages 721–756. Elsevier, 2006.
 - 21 James Brotherston. Cyclic Proofs for First-Order Logic with Inductive Definitions. In Bernhard Beckert, editor, *Proceedings of TABLEAUX 2005*, volume 3702 of *Lecture Notes in Computer Science*, pages 78–92. Springer, 2005. doi:10.1007/11554554_8.
 - 22 James Brotherston and Alex Simpson. Complete Sequent Calculi for Induction and Infinite Descent. In *Proceedings of LICS 2007*, pages 51–62. IEEE Computer Society, 2007. doi:10.1109/LICS.2007.16.
 - 23 Ranald Clouston and Rajeev Goré. Sequent Calculus in the Topos of Trees. In Andrew M. Pitts, editor, *Proc. of FoSSaCS 2015*, volume 9034 of *LNCS*, pages 133–147. Springer, 2015. doi:10.1007/978-3-662-46678-0_9.
 - 24 J. Robin B. Cockett. Deforestation, Program Transformation, and Cut-Elimination. *Electr. Notes Theor. Comput. Sci.*, 44(1):88–127, 2001. doi:10.1016/S1571-0661(04)80904-6.
 - 25 The Coq Development Team. The Coq Proof Assistant Reference Manual. Technical report, LogiCal Project, 2012. Version 8.4. URL: <http://coq.inria.fr>.
 - 26 Christian Dax, Martin Hofmann, and Martin Lange. A Proof System for the Linear Time μ -Calculus. In S. Arun-Kumar and Naveen Garg, editors, *Proceedings of FSTTCS 2006*, volume 4337 of *LNCS*, pages 273–284. Springer, 2006. doi:10.1007/11944836_26.
 - 27 Amina Doumane. *On the Infinitary Proof Theory of Logics with Fixed Points*. PhD Thesis, Université Paris Diderot, 2017.
 - 28 Jérôme Fortier and Luigi Santocanale. Cuts for Circular Proofs: Semantics and Cut-Elimination. In *CSL*, pages 248–262, 2013. doi:10.4230/LIPIcs.CSL.2013.248.
 - 29 Clément Fumex, Neil Ghani, and Patricia Johann. Indexed Induction and Coinduction, Fibrationally. In *Proc. of CALCO '11*, volume 6859 of *Lecture Notes in Computer Science*, pages 176–191. Springer, 2011. doi:10.1007/978-3-642-22944-2_13.
 - 30 Sergey Goncharov and Lutz Schröder. Guarded Traced Categories. In Christel Baier and Ugo Dal Lago, editors, *Proc. of FOSSACS'18*, volume 10803 of *LNCS*, pages 313–330. Springer, 2018. doi:10.1007/978-3-319-89366-2_17.
 - 31 Programming Logic group on Agda. Agda Documentation. Technical report, Chalmers and Gothenburg University, 2015. Version 2.4.2.5. URL: <http://wiki.portal.chalmers.se/agda/>.
 - 32 Tatsuya Hagino. A Typed Lambda Calculus with Categorical Type Constructors. In *Category Theory in Computer Science*, Lecture Notes in Computer Science, pages 140–157. Springer, 1987. doi:10.1007/3-540-18508-9_24.
 - 33 Helle Hvid Hansen, Clemens Kupke, and Jan Rutten. Stream Differential Equations: Specification Formats and Solution Methods. *LMCS*, 13(1), 2017. doi:10.23638/LMCS-13(1:3)2017.
 - 34 Masahito Hasegawa. On Traced Monoidal Closed Categories. *Mathematical Structures in Computer Science*, 19(2):217–244, 2009. doi:10.1017/S0960129508007184.
 - 35 Ichiro Hasuo, Kenta Cho, Toshiki Kataoka, and Bart Jacobs. Coinductive Predicates and Final Sequences in a Fibration. *Electronic Notes in Theoretical Computer Science*, 298:197–214, November 2013. doi:10.1016/j.entcs.2013.09.014.
 - 36 Claudio Hermida and Bart Jacobs. Structural Induction and Coinduction in a Fibrational Setting. *Information and Computation*, 145:107–152, 1997. doi:10.1006/inco.1998.2725.
 - 37 Chris Heunen, Ohad Kammar, Sam Staton, and Hongseok Yang. A Convenient Category for Higher-Order Probability Theory. In *Proc. of LICS'17*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005137.

- 38 Chung-Kil Hur, Georg Neis, Derek Dreyer, and Viktor Vafeiadis. The Power of Parameterization in Coinductive Proof. In *Proc. of POPL'13*, POPL '13, pages 193–206. ACM, 2013. doi:10.1145/2429069.2429093.
- 39 Michael Huth and Marta Z. Kwiatkowska. Quantitative Analysis and Model Checking. In *Proc. of LICS'97*, pages 111–122. IEEE Computer Society, 1997. doi:10.1109/LICS.1997.614940.
- 40 J Martin E Hyland. The Effective Topos. In *Studies in Logic and the Foundations of Mathematics*, volume 110, pages 165–216. Elsevier, 1982.
- 41 Bart Jacobs. *Categorical Logic and Type Theory*. Number 141 in Studies in Logic and the Foundations of Mathematics. North Holland, Amsterdam, 1999.
- 42 Bart Jacobs. *Introduction to Coalgebra: Towards Mathematics of States and Observation*. Number 59 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2016. doi:10.1017/CB09781316823187.
- 43 Bart Jacobs and Jan Rutten. A Tutorial on (Co)Algebras and (Co)Induction. *EATCS Bulletin*, 62:62–222, 1997.
- 44 Henning Kerstan, Barbara König, and Bram Westerbaan. Lifting Adjunctions to Coalgebras to (Re)Discover Automata Constructions. In Marcello M. Bonsangue, editor, *Revised Selected Papers of CMCS'14*, volume 8446 of *LNCS*, pages 168–188. Springer, 2014. doi:10.1007/978-3-662-44124-4_10.
- 45 Anders Kock. Strong Functors and Monoidal Monads. *Archiv der Mathematik*, 23(1):113–120, 1972.
- 46 Dexter Kozen. Results on the Propositional μ -Calculus. *Theor. Comput. Sci.*, 27:333–354, 1983. doi:10.1016/0304-3975(82)90125-6.
- 47 F William Lawvere. Diagonal Arguments and Cartesian Closed Categories. In *Category Theory, Homology Theory and Their Applications II*, pages 134–145. Springer, 1969.
- 48 Nex Paul Mendler. Inductive Types and Type Constraints in the Second-Order Lambda Calculus. *Ann. Pure Appl. Logic*, 51(1-2):159–172, 1991. doi:10.1016/0168-0072(91)90069-X.
- 49 Matteo Mio and Alex Simpson. Łukasiewicz (μ)-calculus. *Fundam. Inform.*, 150(3-4):317–346, 2017. doi:10.3233/FI-2017-1472.
- 50 Rasmus Ejlers Møgelberg. A Type Theory for Productive Coprogramming via Guarded Recursion. In *CSL-LICS*, pages 71:1–71:10. ACM, 2014. doi:10.1145/2603088.2603132.
- 51 Lawrence S. Moss. Parametric Corecursion. *Theoretical Computer Science*, 260:139–163, 2001.
- 52 Hiroshi Nakano. A Modality for Recursion. In *LICS*, pages 255–266. IEEE Computer Society, 2000. doi:10.1109/LICS.2000.855774.
- 53 Damian Niwinski and Igor Walukiewicz. Games for the μ -Calculus. *TCS*, 163(1&2):99–116, 1996. doi:10.1016/0304-3975(95)00136-0.
- 54 Damien Pous. Complete Lattices and Up-To Techniques. In Zhong Shao, editor, *APLAS'07*, volume 4807 of *LNCS*, pages 351–366. Springer, 2007. doi:10.1007/978-3-540-76637-7_24.
- 55 Damien Pous. Coinduction All the Way Up. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of LICS '16*, pages 307–316. ACM, 2016. doi:10.1145/2933575.2934564.
- 56 Damien Pous and Jurriaan Rot. Companions, Codensity, and Causality. In *Proceedings of FOSSACS 2017*, 2017. doi:10.1007/978-3-662-54458-7_7.
- 57 Grigore Roşu and Dorel Lucanu. Circular Coinduction: A Proof Theoretical Foundation. In *CALCO*, volume 5728 of *LNCS*, pages 127–144. Springer, 2009. doi:10.1007/978-3-642-03741-2_10.
- 58 Jurriaan Rot. *Enhanced Coinduction*. PhD, University Leiden, Leiden, 2015.
- 59 Jurriaan Rot, Filippo Bonchi, Marcello Bonsangue, Damien Pous, Jan Rutten, and Alexandra Silva. Enhanced Coalgebraic Bisimulation. *MSCS*, 27(7):1236–1264, 2017. doi:10.1017/S0960129515000523.
- 60 Jan Rutten. Universal Coalgebra: A Theory of Systems. *TCS*, 249(1):3–80, 2000. doi:10.1016/S0304-3975(00)00056-6.

- 61 Jan Rutten. Behavioural Differential Equations: A Coinductive Calculus of Streams, Automata, and Power Series. *TCS*, 308(1-3):1–53, 2003. doi:10.1016/S0304-3975(02)00895-2.
- 62 Jan Rutten. *The Method of Coalgebra: Exercises in Coinduction*. CWI, Amsterdam, February 2019. URL: <http://persistent-identifier.org/?identifier=urn:nbn:nl:ui:18-28550>.
- 63 Davide Sangiorgi. On the Bisimulation Proof Method. *Mathematical Structures in Computer Science*, 8(5):447–479, 1998.
- 64 Luigi Santocanale. A Calculus of Circular Proofs and Its Categorical Semantics. In *FoSSaCS*, pages 357–371, 2002. doi:10.1007/3-540-45931-6_25.
- 65 Luigi Santocanale. μ -Bicomplete Categories and Parity Games. *RAIRO - ITA*, 36(2):195–227, 2002. doi:10.1051/ita:2002010.
- 66 Luke Simon, Ajay Bansal, Ajay Mallya, and Gopal Gupta. Co-Logic Programming: Extending Logic Programming with Coinduction. In Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki, editors, *Proc. of ICALP'07*, volume 4596 of *LNCS*, pages 472–483. Springer, 2007. doi:10.1007/978-3-540-73420-8_42.
- 67 Alex Simpson. Cyclic Arithmetic Is Equivalent to Peano Arithmetic. In *Proceedings of FoSSaCS'17*, LNCS, 2017. doi:10.1007/978-3-662-54458-7_17.
- 68 Craig Smoryński. *Self-Reference and Modal Logic*. Universitext. Springer-Verlag, 1985.
- 69 Ana Sokolova. Probabilistic Systems Coalgebraically: A Survey. *TCS*, 412(38):5095–5110, 2011. doi:10.1016/j.tcs.2011.05.008.
- 70 Robert M. Solovay. Provability Interpretations of Modal Logic. *Israel Journal of Mathematics*, 25(3):287–304, 1976. doi:10.1007/BF02757006.
- 71 Thomas Streicher. Fibred Categories à La Jean Bénabou. *arXiv:math.CT/1801.02927*, 2018. arXiv:1801.02927.
- 72 Kasper Svendsen, Filip Sieczkowski, and Lars Birkedal. Transfinite Step-Indexing: Decoupling Concrete and Logical Steps. In Peter Thiemann, editor, *Proc. of ESOP'16*, volume 9632 of *LNCS*, pages 727–751. Springer, 2016. doi:10.1007/978-3-662-49498-1_28.
- 73 D. A. Turner. Elementary Strong Functional Programming. In Pieter H. Hartel and Marinus J. Plasmeijer, editors, *Proceedings of FPLE'95*, volume 1022 of *LNCS*, pages 1–13. Springer, 1995. doi:10.1007/3-540-60675-0_35.
- 74 Matthijs Vákár, Ohad Kammar, and Sam Staton. A Domain Theory for Statistical Probabilistic Programming. *PACMPL*, 3(POPL):36:1–36:29, 2019. arXiv:1811.04196.
- 75 Igor Walukiewicz. On Completeness of the Mu-Calculus. In *Proceedings of LICS '93*, pages 136–146. IEEE Computer Society, 1993. doi:10.1109/LICS.1993.287593.

A Interpretation of the Probabilistic Modal μ -Calculus

Given a formula φ with no or one free variable X , a Segala system $c: Q \rightarrow \mathcal{S}(Q)$ and an interpretation $I: Q \rightarrow \text{qPred}_{\text{At}}$, we use Theorem 30 to define a locally contractive functor $\llbracket \varphi \rrbracket: \overline{\text{Pred}}_Q^n \rightarrow \overline{\text{Pred}}_Q$ with $n = 0, 1$, where $\heartsuit_Q = \top_Q, \wedge_Q, \dots$ are the corresponding fibred connectives in $\overline{\text{qPred}}_Q$:

$$\begin{array}{lll}
\llbracket P \rrbracket = K(I(P)) & \llbracket \overline{P} \rrbracket = K(1 - I(P)) & \llbracket X_k \rrbracket = \blacktriangleright \circ \pi_k \\
\llbracket \top \rrbracket = \top_Q & \llbracket \perp \rrbracket = \perp_Q & \llbracket \nu X. \varphi \rrbracket = \nu \llbracket \varphi \rrbracket \\
\llbracket \blacktriangleright \varphi \rrbracket = \blacktriangleright \circ \llbracket \varphi \rrbracket & \llbracket \square \varphi \rrbracket = c^\# \circ \overleftarrow{\mathcal{S}^\square} \circ \llbracket \varphi \rrbracket & \llbracket \diamond \varphi \rrbracket = c^\# \circ \overleftarrow{\mathcal{S}^\diamond} \circ \llbracket \varphi \rrbracket \\
\llbracket \varphi \sqcap \psi \rrbracket = \llbracket \varphi \rrbracket \wedge_Q \llbracket \psi \rrbracket & \llbracket \varphi \sqcup \psi \rrbracket = \llbracket \varphi \rrbracket \vee_Q \llbracket \psi \rrbracket & \llbracket \varphi \rightarrow \psi \rrbracket = \llbracket \varphi \rrbracket \Rightarrow_Q \llbracket \psi \rrbracket
\end{array}$$

B

 Types and Terms for Guarded Probabilistic Programming

Type, context and term formation rules for guarded probabilistic programming:

$$\begin{array}{c}
\frac{X \in \Delta}{\Delta \Vdash X : \mathbf{Ty}} \quad \frac{\Delta \Vdash A : \mathbf{Ty}}{\Delta \Vdash \blacktriangleright A : \mathbf{Ty}} \quad \frac{\Delta, X \Vdash A : \mathbf{Ty} \quad X \text{ appears under } \blacktriangleright \text{ in } A}{\Delta \Vdash \text{fix } X. A : \mathbf{Ty}} \\
\frac{}{\Delta \Vdash \mathbb{R} : \mathbf{Ty}} \quad \frac{\Delta \Vdash A : \mathbf{Ty} \quad \Delta \Vdash B : \mathbf{Ty}}{\Delta \Vdash A \times B : \mathbf{Ty}} \quad \frac{\Delta \Vdash A : \mathbf{Ty} \quad \Delta \Vdash B : \mathbf{Ty}}{\Delta \Vdash A \rightarrow B : \mathbf{Ty}} \\
\frac{}{\cdot \text{Ctx}} \quad \frac{\Gamma \text{Ctx} \quad x \notin \Gamma \quad \Delta \Vdash B : \mathbf{Ty}}{\Gamma \text{Ctx}} \\
\frac{x : A \in \Gamma}{\Gamma \vdash x : A} \quad \frac{\Gamma \vdash t : A}{\Gamma \vdash \text{next } t : \blacktriangleright A} \quad \frac{\Gamma \vdash t : \blacktriangleright (A \rightarrow B) \quad \Gamma \vdash s : \blacktriangleright A}{\Gamma \vdash t \otimes s : \blacktriangleright B} \\
\frac{\Gamma, x : \blacktriangleright A \vdash t : A}{\Gamma \vdash \text{fix } x. t : A} \quad \frac{\Gamma \vdash t : A[\text{fix } X. A/X]}{\Gamma \vdash \text{in } t : \text{fix } X. A} \quad \frac{\Gamma \vdash t : \text{fix } X. A}{\Gamma \vdash \text{out } t : A[\text{fix } X. A/X]} \\
\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x. t : A \rightarrow B} \quad \frac{\Gamma \vdash t : A \rightarrow B \quad \Gamma \vdash s : A}{\Gamma \vdash t s : B} \\
\frac{\Gamma \vdash t : A \quad \Gamma \vdash s : B}{\Gamma \vdash \langle t, s \rangle : A \times B} \quad \frac{\Gamma \vdash t : A \times B}{\Gamma \vdash \text{fst } t : A} \quad \frac{\Gamma \vdash t : A \times B}{\Gamma \vdash \text{snd } t : B} \\
\frac{a \in \mathbb{Q}}{\Gamma \vdash a : \mathbb{R}} \quad \frac{}{\Gamma \vdash \text{normal} : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}}
\end{array}$$

Before we come to the semantics, let us single out values. This will simplify the denotational semantics, as values need to be embedded into the monad \overleftarrow{P} that we will use to model the probabilistic effects of the language.

$$v, w, u ::= x \mid a \mid \langle v, w \rangle \mid \text{in } v \mid \lambda x. t$$

We denote by Tm_A^Γ the terms of type A in context Γ and by Val_A^Γ the values of type A .

Let $P : \mathbf{qBS} \rightarrow \mathbf{qBS}$ be the strong probability monad on quasi-Borel spaces, see [74]. As monads lift easily point-wise to descending chain, we get a strong monad \overleftarrow{P} on $\overleftarrow{\mathbf{qBS}}$ with unit $\overleftarrow{\eta}$ and bind operator $\overleftarrow{\gg}$. Note that the unit and multiplication of the monad lift point-wise, while the strength needs to be defined through the unique mapping property of the end that we used to construct exponentials in $\overleftarrow{\mathbf{qBS}}$. In \mathbf{qBS} , we also find the normal distribution given as morphism $N : \mathbb{R} \times \mathbb{R} \rightarrow P(\mathbb{R})$. This gives us a morphism $K(N) : K(\mathbb{R}) \times K(\mathbb{R}) \rightarrow \overleftarrow{P}(K(\mathbb{R}))$, which we will use below. Next, we need a natural transformation $\iota : \blacktriangleright \overleftarrow{P} \Rightarrow \overleftarrow{P} \blacktriangleright$, which is given by $\iota_{\sigma,0} = \mathbf{1} \cong P\mathbf{0} \xrightarrow{P!} P\mathbf{1}$ and $\iota_{\sigma,n+1} = \text{id}_{P\sigma_n}$.

The interpretation of types, context, values and terms over $\overleftarrow{\mathbf{qBS}}$ is then given as follows.

$$\begin{aligned}
\llbracket \Delta \Vdash A : \mathbf{Ty} \rrbracket &: \overleftarrow{\mathbf{qBS}}^\Delta \rightarrow \overleftarrow{\mathbf{qBS}} \\
\llbracket \Gamma \text{Ctx} \rrbracket &\in \overleftarrow{\mathbf{qBS}} \\
\llbracket - \rrbracket^{\text{val}} &: \text{Val}_A^\Gamma \rightarrow \overleftarrow{\mathbf{qBS}}(\llbracket \Gamma \rrbracket, \llbracket A \rrbracket) \\
\llbracket - \rrbracket &: \text{Tm}_A^\Gamma \rightarrow \overleftarrow{\mathbf{qBS}}(\llbracket \Gamma \rrbracket, \overleftarrow{P}\llbracket A \rrbracket)
\end{aligned}$$

8:22 Coinduction in Flow

$$\begin{array}{ll}
\llbracket X \rrbracket = \pi_X & \llbracket \blacktriangleright A \rrbracket = \blacktriangleright \circ \llbracket A \rrbracket \\
\llbracket \text{fix } X. A \rrbracket = \nu \llbracket A \rrbracket & \llbracket \mathbb{R} \rrbracket = K(\mathbb{R}) \\
\llbracket A \times B \rrbracket = \llbracket A \rrbracket \times \llbracket B \rrbracket & \llbracket A \rightarrow B \rrbracket = \llbracket A \rrbracket \Rightarrow \overleftarrow{P} \llbracket B \rrbracket
\end{array}$$

$$\llbracket \cdot \rrbracket = \mathbf{1} \qquad \llbracket \Gamma, x : A \rrbracket = \llbracket \Gamma \rrbracket \times \llbracket A \rrbracket$$

$$\begin{array}{ll}
\llbracket x \rrbracket^{\text{val}} = \pi_x & \llbracket a \rrbracket^{\text{val}} = \lambda \gamma. K(a) \\
\llbracket \langle v, w \rangle \rrbracket^{\text{val}} = \langle \llbracket v \rrbracket^{\text{val}}, \llbracket w \rrbracket^{\text{val}} \rangle & \llbracket \text{in } v \rrbracket^{\text{val}} = \xi^{-1} \circ \llbracket v \rrbracket^{\text{val}} \\
\llbracket \lambda x. t \rrbracket^{\text{val}} = \lambda \llbracket t \rrbracket
\end{array}$$

$$\begin{array}{ll}
\llbracket v \rrbracket \gamma = \overleftarrow{\eta}_{\llbracket A \rrbracket}(\llbracket v \rrbracket^{\text{val}} \gamma) & \llbracket \text{next } t \rrbracket \gamma = \overleftarrow{P}(\text{next}_{\llbracket A \rrbracket})(\llbracket t \rrbracket \gamma) \\
\llbracket t \otimes s \rrbracket \gamma = \llbracket t \rrbracket \gamma \overleftarrow{\otimes} \lambda f. \llbracket s \rrbracket \gamma \overleftarrow{\otimes} \lambda x. \text{mon } f \ x & \llbracket \text{fix } x. t \rrbracket \gamma = \text{löb}(\lambda x. \iota(x) \overleftarrow{\otimes} \lambda y. \llbracket t \rrbracket(\gamma, y)) \\
\llbracket \text{in } t \rrbracket \gamma = \overleftarrow{P}(\xi^{-1})(\llbracket t \rrbracket \gamma) & \llbracket \text{out } t \rrbracket \gamma = \overleftarrow{P}(\xi)(\llbracket t \rrbracket \gamma) \\
\llbracket \text{normal} \rrbracket \gamma = \lambda K(N) & \llbracket t \ s \rrbracket \gamma = \llbracket t \rrbracket \gamma \overleftarrow{\otimes} \lambda f. \llbracket s \rrbracket \gamma \overleftarrow{\otimes} \lambda x. f \ x \\
\llbracket \langle t, s \rangle \rrbracket \gamma = \llbracket t \rrbracket \gamma \overleftarrow{\otimes} \lambda x. \llbracket s \rrbracket \gamma \overleftarrow{\otimes} \lambda y. \eta(x, y) & \llbracket \text{fst } t \rrbracket \gamma = \overleftarrow{P}(\pi_1)(\llbracket t \rrbracket \gamma) \\
\llbracket \text{snd } t \rrbracket \gamma = \overleftarrow{P}(\pi_2)(\llbracket t \rrbracket \gamma)
\end{array}$$

Causal Unfoldings

Marc de Visme

Univ Lyon, ENS de Lyon, CNRS, UCB Lyon 1, LIP, France

Glynn Winskel

Computer Laboratory, University of Cambridge, UK

Abstract

In the simplest form of event structure, a *prime* event structure, an event is associated with a unique causal history, its prime cause. However, it is quite common for an event to have disjunctive causes in that it can be enabled by any one of multiple sets of causes. Sometimes the sets of causes may be mutually exclusive, inconsistent one with another, and sometimes not, in which case they coexist consistently and constitute *parallel* causes of the event. The established model of *general* event structures can model parallel causes. On occasion however such a model abstracts too far away from the precise causal histories of events to be directly useful. For example, sometimes one needs to associate probabilities with different, possibly coexisting, causal histories of a common event. Ideally, the causal histories of a general event structure would correspond to the configurations of its *causal unfolding* to a prime event structure; and the causal unfolding would arise as a right adjoint to the embedding of prime in general event structures. But there is no such adjunction. However, a slight extension of prime event structures remedies this defect and provides a causal unfolding as a universal construction. Prime event structures are extended with an equivalence relation in order to dissociate the two roles, that of an event and its enabling; in effect, prime causes are labelled by a disjunctive event, an equivalence class of its prime causes. With this enrichment a suitable causal unfolding appears as a pseudo right adjoint. The adjunction relies critically on the central and subtle notion of *extremal causal realisation* as an embodiment of causal history.

2012 ACM Subject Classification Theory of computation → Concurrency

Keywords and phrases Event Structures, Parallel Causes, Causal Unfolding, Probability

Digital Object Identifier 10.4230/LIPIcs.CALCO.2019.9

Acknowledgements Thanks to the anonymous referees. Thanks to Simon Castellan, Pierre Clairambault, Ioana Cristescu, Mai Gehrke, Jonathan Hayman, Tamas Kispeter, Jean Krivine, Martin Hyland and Daniele Varacca for discussions, advice and encouragement; to ENS Paris for supporting Marc de Visme’s internship; and to the ERC for Advanced Grant ECSYM.

1 Introduction

Work on probabilistic distributed strategies based on event structures brought us face to face with a limitation in existing models of concurrent computation, and in particular with the theory of event structures as it had been developed. In order to adequately express certain intuitively natural, optimal probabilistic strategies, it was necessary to simultaneously support: probability on event structures with opponent moves, itself rather subtle; parallel causes, in which an event may be enabled in several distinct but compatible ways; and a hiding operation crucial in the composition of strategies. The difficulties did not show up in the less refined development of nondeterministic strategies; there the simplest form of event structure, prime event structures, sufficed. The “obvious” remedy, to base strategies on more general event structures, which do support parallel causes, failed to support probability and hiding adequately. The problems and a solution are documented in the article [7].

That work uncovered a central construction, which we here call the *causal unfolding* of a model with parallel causes. It is based on the notion of *extremal causal realisation* and attendant *prime extremal realisation* which plays a role analogous to that of complete



© Marc de Visme and Glynn Winskel;

licensed under Creative Commons License CC-BY

8th Conference on Algebra and Coalgebra in Computer Science (CALCO 2019).

Editors: Markus Roggenbach and Ana Sokolova; Article No. 9; pp. 9:1–9:18

Leibniz International Proceedings in Informatics

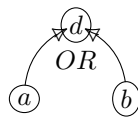


LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

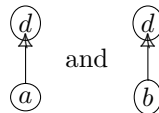
9:2 Causal Unfoldings

prime in distributive orders. Both concepts deserve to be better known and are expanded on comprehensively with full proofs here. As will shortly be explained more fully, intuitively, a prime extremal realisation is a finite partial order expressing a minimal causal history for an event to occur, even in the presence of several parallel causes for the event. Extremal realisations provide us with a way to unfold a model supporting parallel causes (general event structures – Section 2.2, or equivalence families – Section 3) into a structure describing all its causal histories – its causal unfolding. As is to be hoped, the unfolding will be a form of right adjoint giving the causal unfolding and extremal realisations a categorical significance.

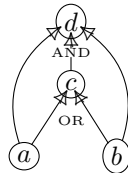
To give an idea of prime extremal realisations of events we give a short, necessarily informal, preview of two examples from the paper. The simplest concerns a general event structure comprising three events a , b and d where d can occur once a or b have occurred and where all events can occur together. The events a and b constitute parallel causes of the event d . We can picture the situation in the diagram:



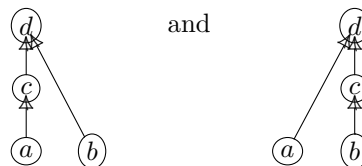
Here there are two minimal causal histories associated with the occurrence of the event d , viz. d after a , and d after b :



These will be the prime extremal realisations associated with the occurrence of d . But this example is deceptively simple. To add a level of difficulty, consider the general event structure



which portrays an event d enabled through the occurrence of all of the events a , b and c but where c is enabled by either a or b . This time the two minimal causal histories associated with the occurrence of the event d , one after c caused by a , and the other after c caused by b , give rise to the two prime extremal realisations:



There are also more subtle “non-injective” prime extremal realisations in which the same event of a general event structure occurs in several different ways – see Example 13, though these have been ruled out in our application to strategies with parallel causes [7].

The new adjunction, with its right adjoint the causal unfolding, supplies a missing link in the landscape of models for concurrency [15]. The adjunction connects models with parallel causes, such as general event structures, to those based on partial orders of events. It does this through the introduction of a simple, new model which is based on prime event structures extended with an equivalence relation on their sets of events.

In systems with parallel causes it is often necessary to associate probabilities with causal histories, and the causal unfolding provides a suitable structure on which to do this systematically [7]. Outside probability, there is a similar need for causal unfoldings, for example, when reversible computing encounters parallel causes [3, 4], and in extracting biochemical pathways, forms of causal history in biochemical systems where parallel causes are rife [5].

2 Event structures and their maps

We briefly review two well-established forms of event structure and explain the absence of an adjunction associated with the embedding of prime into general event structures. It is through such an adjunction one might otherwise have thought to find a causal unfolding of general event structures to prime event structures. The absence motivates a new model based on prime event structures with an equivalence relation. (We refer the reader to [13, 14] in particular for background and intuitions.)

2.1 Prime event structures

A *prime event structure* comprises (E, \leq, Con) , consisting of a set E of *events* which are partially ordered by \leq , the *causal dependency relation*, and a non-empty *consistency relation* Con consisting of finite subsets of E . The relation $e' \leq e$ expresses that event e causally depends on the previous occurrence of event e' . Write $[X]$ for the \leq -down-closure of a subset of events X . That a finite subset of events is consistent conveys that its events can occur together by some stage in the evolution of the process. Together the relations satisfy several axioms:

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

A *configuration* is a, possibly infinite, set of events $x \subseteq E$ which is: *consistent*, $X \subseteq x$ and X is finite implies $X \in \text{Con}$; and *down-closed*, $[x] = x$. It is part and parcel of prime event structures that an event e is associated with a unique causal history $[e]$.

Prime event structures have a long history. They first appeared in describing the patterns of event occurrences that occurred in the unfolding of a (1-safe) Petri net [10]. As their configurations, ordered by inclusion, form a Scott domain, prime event structures provided an early bridge between the semantic theories of Dana Scott and Carl Petri; one early result being that a *confusion-free* Petri net unfolded to a prime event structure with configurations taking the form of a *concrete* domain, as defined by Kahn and Plotkin. Generally, the configurations of a countable prime event structure ordered by inclusion coincide with the dI-domains of Berry – distributed Scott domains which satisfy a finiteness axiom [14]. The domains of configuration of a prime event structure had been characterised earlier in [10] as *prime algebraic* domains, Scott domains with a subbasis of complete primes.¹

¹ A *complete prime* in an order which supports least upper bounds $\bigsqcup X$ of compatible subsets X is an element p such that $p \sqsubseteq \bigsqcup X$ implies $p \sqsubseteq x$ for some $x \in X$. In the configurations of a prime event structure the complete primes are exactly those configurations $[e]$ for an event e .

2.2 General event structures

A *general event structure* [13, 14] permits an event to be caused disjunctively in several ways, possibly coexisting in parallel, as parallel causes. A general event structure comprises (E, Con, \vdash) where E is a set of events, the consistency relation Con is a non-empty collection of finite subsets of E , and the *enabling relation* \vdash is a relation in $\text{Con} \times E$ such that

$$\begin{aligned} X \subseteq Y \in \text{Con} &\implies X \in \text{Con}, \text{ and} \\ Y \in \text{Con} \ \& \ Y \supseteq X \ \& \ X \vdash e &\implies Y \vdash e. \end{aligned}$$

A *configuration* is a subset x of E which is: *consistent*, $X \subseteq_{\text{fin}} x \implies X \in \text{Con}$; and *secured*, $\forall e \in x \exists e_1, \dots, e_n \in x. e_n = e \ \& \ \forall i \leq n. \{e_1, \dots, e_{i-1}\} \vdash e_i$. We write $\mathcal{C}^\infty(E)$ for the configurations of E and $\mathcal{C}(E)$ for its finite configurations. (For illustrations of small general event structure see, for instance, Example 1 and E_0 of Example 12.)

An event e being enabled in a configuration has been expressed through the existence of a securing chain e_1, \dots, e_n , with $e_n = e$, within the configuration. The chain represents a *complete enabling* of e in the sense that every event in the chain is itself enabled by earlier members of the chain. Just as mathematical proofs are most usefully viewed not merely as sequences, so later complete enableings expressed more generally as partial orders – “causal realisations” – will play a central role.

A *map* $f : (E, \text{Con}, \vdash) \rightarrow (E', \text{Con}', \vdash')$ of general event structures is a partial function $f : E \rightarrow E'$ such that

$$\begin{aligned} \forall X \in \text{Con}. fX \in \text{Con}', \\ \forall X \in \text{Con}, e_1, e_2 \in X. f(e_1) = f(e_2) \text{ (both defined)} &\implies e_1 = e_2, \text{ and} \\ \forall X \in \text{Con}, e \in E. X \vdash e \ \& \ f(e) \text{ is defined} &\implies fX \vdash' f(e). \end{aligned}$$

Maps compose as partial functions. Write \mathcal{G} for the category of general event structures.

W.r.t. a family of subsets \mathcal{F} , a subset X of \mathcal{F} is *compatible* (in \mathcal{F}), written $X \uparrow$, if there is $y \in \mathcal{F}$ such that $x \subseteq y$ for all $x \in X$; we write $x \uparrow y$ for $\{x, y\} \uparrow$. Say a subset is *finitely compatible* iff every finite subset is compatible.

We can now characterise those families of configurations arising from a general event structure [14]. A *family of configurations* comprises a non-empty family \mathcal{F} of sets such that if $X \subseteq \mathcal{F}$ is finitely compatible in \mathcal{F} then $\bigcup X \in \mathcal{F}$; and if $e \in x \in \mathcal{F}$ there is a securing chain $e_1, \dots, e_n = e$ in x such that $\{e_1, \dots, e_i\} \in \mathcal{F}$ for all $i \leq n$.² Its *events* are elements of the underlying set $\bigcup \mathcal{F}$. A *map* between families of configurations from \mathcal{A} to \mathcal{B} is a partial function $f : \bigcup \mathcal{A} \rightarrow \bigcup \mathcal{B}$ between their events such that $fx \in \mathcal{B}$ if $x \in \mathcal{A}$ and any event of fx arises as the image of a unique event of x . Maps compose as partial functions. Write \mathcal{Fam} for the category of families of configurations.

Characterisations of the orders obtained from the configurations of a general event structure can be found in [13].³

² The latter condition is equivalent to: (i) if $e \in x \in \mathcal{F}$ there is a finite $x_0 \in \mathcal{F}$ s.t. $e \in x_0 \in \mathcal{F}$ and (ii) (coincident-freeness) for distinct $e, e' \in x$, there is $y \in \mathcal{F}$ with $y \subseteq x$ s.t. $e \in y \iff e' \notin y$.

³ Complete irreducibles are the customary generalisation of complete primes to nondistributive orders such as those of configurations of general event structures ordered by inclusion [13]. A *complete irreducible* in an order which supports least upper bounds $\bigsqcup X$ of compatible subsets X is an element r such that $r = \bigsqcup X$ implies $r = x$ for some $x \in X$. In the configurations of a general event structure the complete irreducibles are exactly those minimal configurations which contain an event e . A forewarning: only in very special circumstances will prime extremal realisations – the generalisation of complete prime of this paper – coincide with complete irreducibles – see Example 12.

2.3 A coreflection and non-coreflection

There is a forgetful functor $\mathcal{G} \rightarrow \mathcal{Fam}$ taking a general event structure to its family of configurations. It has a left adjoint, which constructs a canonical general event structure from a family: given \mathcal{A} , a family of configurations with underlying events A , construct a general event structure (A, Con, \vdash) with $X \in \text{Con}$ iff $X \subseteq_{\text{fin}} y$, for some $y \in \mathcal{A}$; and with $X \vdash a$ iff $a \in A$, $X \in \text{Con}$ and $a \in y \subseteq X \cup \{a\}$, for some $y \in \mathcal{A}$.

The above yields a coreflection⁴

$$\mathcal{Fam} \begin{array}{c} \longleftarrow \\ \top \\ \longrightarrow \end{array} \mathcal{G}$$

of families of configurations in general event structures. It cuts down to an equivalence between families of configurations and *replete* general event structures. A general event structure (E, Con, \vdash) is *replete* iff

$$\begin{aligned} \forall e \in E \exists X \in \text{Con}. X \vdash e, \quad \forall X \in \text{Con} \exists x \in \mathcal{C}(E). X \subseteq x \text{ and} \\ X \vdash e \implies \exists x \in \mathcal{C}(E). e \in x \ \& \ x \subseteq X \cup \{e\}. \end{aligned}$$

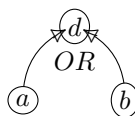
A *map* of prime event structures is a map of their families of configurations. Write \mathcal{E} for the category of prime event structures. (A map in \mathcal{E} need not preserve causal dependency; when it does and is total it is called *rigid*.)

There is an obvious “inclusion” functor $\mathcal{E} \rightarrow \mathcal{Fam}$ fully and faithfully embedding the category of prime event structures in the category of families of configurations and so in general event structures. We might expect the functor $\mathcal{E} \rightarrow \mathcal{Fam}$ to be the left adjoint of a coreflection

$$\mathcal{E} \begin{array}{c} \overset{?}{\longleftarrow} \\ \top \\ \longrightarrow \end{array} \mathcal{Fam} \begin{array}{c} \longleftarrow \\ \top \\ \longrightarrow \end{array} \mathcal{G},$$

so yielding a composite right adjoint $\mathcal{G} \rightarrow \mathcal{E}$ which unfolds a general event structure to a prime event structure [14, 15]. However under reasonable assumptions this cannot exist, as the following example indicates.

► **Example 1.** Consider a general event structure comprising three events a , b and d with all subsets consistent and minimal enablings $\emptyset \vdash a, b$ and $\{a\} \vdash d$ and $\{b\} \vdash d$. Imagine concurrent treatments a and b of two doctors which sadly lead to the death d of the patient.



As its unfolding it is hard to avoid a prime event structure with events and causal dependency $a < d_a$ and $b < d_b$ – the event d_a representing “death by a ” and the event d_b “death by b ” – with the counit of the adjunction collapsing d_a and d_b to the common event d . (If we are to apportion blame to the doctors we shall need the probabilities of d_a and d_b given a and b [11].) In order for the counit to be a map we are forced to make $\{d_a, d_b\}$ inconsistent. This is one issue: why should death by one doctor’s treatment be in conflict with death by the other’s – they could be jointly responsible? But even more damningly the tentative counit fails the universal property required of it! Consider another prime event structure with three events comprising $a < d$ and $b < d$ (“death due to both doctors’ treatments”). The obvious

⁴ A coreflection is an adjunction where the left adjoint is full and faithful, or equivalently the unit is iso.

map to the family of configurations of the general event structure – the identity on events – fails to factor *uniquely* through the putative counit: d can be sent to either d_a or d_b ; the event “death by both doctors” can be sent to either “death by a” or “death by b.” This raises the second issue: if we are to obtain the required universal property we have to regard these two maps as essentially the same.

The two issues raised in the example suggest a common solution: to enrich prime event structures with equivalence relations. This will allow a broader class of maps, settling the first issue, and introduce an equivalence on maps, settling the second. The causal unfolding of the “doctors example” will be very simple and comprise the prime event structure $a < d_a$ and $b < d_b$ with d_a and d_b equivalent events; with all events consistent. In general the construction of the unfolding is surprisingly involved; causal histories can be much more intricate than in the simple example.

3 Events with an equivalence, categories \mathcal{E}_{\equiv} and \mathcal{Fam}_{\equiv}

We build causal unfoldings in a new model, based on the obvious extension to events with an equivalence relation. A (*prime*) *event structure with equivalence* (an ese) is a structure

$$(P, \leq, \text{Con}, \equiv)$$

where (P, \leq, Con) satisfies the axioms of a prime event structure and \equiv is an equivalence relation on P . The intention is that the events of P represent *prime causes* while the \equiv -equivalence classes of P represent *disjunctive events*: p in P is a prime cause of the event $\{p\}_{\equiv}$. Notice there may be several prime causes of the same event and that these may be parallel causes in the sense that they are consistent with each other and causally independent.

The extension by an equivalence relation on events is accompanied by an extension to families of configurations. An *equivalence-family* (ef) is a family of configurations \mathcal{A} with an equivalence relation \equiv_A on its underlying set $A =_{\text{def}} \bigcup \mathcal{A}$ (with no further axioms). Equivalence-families are the most general model we shall consider; they support parallel causes and, later, a causal unfolding.

Let (\mathcal{A}, \equiv_A) and (\mathcal{B}, \equiv_B) be ef's, with respective underlying sets A and B . A map $f : (\mathcal{A}, \equiv_A) \rightarrow (\mathcal{B}, \equiv_B)$ is a partial function $f : A \rightarrow B$ which preserves \equiv , if $a_1 \equiv_A a_2$ then either both $f(a_1)$ and $f(a_2)$ are undefined or both defined with $f(a_1) \equiv_B f(a_2)$, such that

$$x \in \mathcal{A} \implies fx \in \mathcal{B} \ \& \ \forall a_1, a_2 \in x. f(a_1) \equiv_B f(a_2) \implies a_1 \equiv_A a_2.$$

Composition is composition of partial functions. We regard two maps

$$f_1, f_2 : (\mathcal{A}, \equiv_A) \rightarrow (\mathcal{B}, \equiv_B)$$

as equivalent, and write $f_1 \equiv f_2$, iff they are equidefined and yield equivalent results, *i.e.* if $f_1(p)$ is defined then so is $f_2(p)$ and $f_1(p) \equiv_Q f_2(p)$, and if $f_2(p)$ is defined then so is $f_1(p)$ and $f_1(p) \equiv_Q f_2(p)$. Composition respects \equiv . This yields a category of equivalence families \mathcal{Fam}_{\equiv} ; it is enriched in the category of sets with equivalence relations (also called setoids).⁵

Clearly from an ese (P, \equiv_P) we obtain an ef $(\mathcal{C}^\infty(P), \equiv_P)$ and we take a map of ese's to be a map between their associated ef's. Write \mathcal{E}_{\equiv} for the category of ese's; it too is enriched in the category of sets with equivalence relations. When the equivalence relations \equiv of ese's are

⁵ The Appendix provides background in categories enriched in equivalence relations.

the identity we essentially have prime event structures and their maps. There is clearly a full-and-faithful embedding

$$\mathcal{E}_{\equiv} \rightarrow \mathcal{Fam}_{\equiv},$$

which preserves and reflects the equivalence on maps. One virtue of ese's is that they support a hiding operation, associated with a factorisation system [7].

We sometimes use an alternative description of their maps:

► **Proposition 2.** *A map of ese's from P to Q is a partial function $f : P \rightarrow Q$ which preserves \equiv such that*

- (i) *for all $X \in \text{Con}_P$ the direct image $fX \in \text{Con}_Q$ and $\forall p_1, p_2 \in X. f(p_1) \equiv_Q f(p_2) \implies p_1 \equiv_P p_2$, and*
- (ii) *whenever $q \leq_Q f(p)$ there is $p' \leq_P p$ such that $f(p') = q$.*

While an ese determines an ef, the converse, how to construct the causal unfolding of an ef to an ese, is much less clear. To do so we follow up on the idea of Section 2.2 of basing minimal complete enablings on partial orders. A minimal complete enabling will correspond to a *prime extremal realisation*. Realisations and extremal realisations are our next topic.

4 Causal histories as extremal realisations

Extremal causal realisations formalise the notion of causal history in models with parallel causes, *viz.* general event structures and the most general model of equivalence-families. They will be the central tool in constructing the causal unfoldings of such models.

4.1 Causal realisations

Let \mathcal{A} be a family of configurations with underlying set A . A (*causal*) *realisation* of \mathcal{A} comprises a partial order (E, \leq) , its *carrier*, such that the set $\{e' \in E \mid e' \leq e\}$ is finite for all events $e \in E$, together with a function $\rho : E \rightarrow A$ for which the image $\rho x \in \mathcal{A}$ when x is a down-closed subset of E . We say a realisation is *injective* when it is injective as a function.

A map between realisations $(E, \leq), \rho$ and $(E', \leq'), \rho'$ is a partial surjective function $f : E \rightarrow E'$ which preserves down-closed subsets and satisfies $\rho(e) = \rho'(f(e))$ for all $e \in E$ where $f(e)$ is defined. It is convenient to write such a map as $\rho \succeq^f \rho'$. Occasionally we shall write $\rho \succeq \rho'$, or the converse $\rho' \preceq \rho$, to mean there is a map of realisations from ρ to ρ' .

A map of realisations $\rho \succeq^f \rho'$ factors into a “projection” followed by a total map

$$\rho \succeq_1^{f_1} \rho_0 \succeq_2^{f_2} \rho',$$

where ρ_0 stands for the realisation $(E_0, \leq_0), \rho_0$ where $E_0 = \{e \in E \mid f(e) \text{ is defined}\}$ is the domain of definition of f ; \leq_0 is the restriction of \leq ; f_1 is the inverse relation to the inclusion $E_0 \subseteq E$; and $f_2 : E_0 \rightarrow E'$ is the total part of function f . We are using \succeq_1 and \succeq_2 to signify the two kinds of maps. Notice that \succeq_1 -maps are reverse inclusions. Notice too that \succeq_2 -maps are exactly the total maps of realisations. Total maps $\rho \succeq_2^f \rho'$ are precisely those functions f from the carrier of ρ to the carrier of ρ' which preserve down-closed subsets and satisfy $\rho = \rho' f$.

4.2 Extremal realisations

Let \mathcal{A} be a configuration family with underlying set A . We shall say a realisation ρ is *extremal* when $\rho \succeq_2^f \rho'$ implies f is an isomorphism, for any realisation ρ' ; it is called *prime extremal* when it in addition has a top element, *i.e.* its carrier contains an element which dominates all other elements in the carrier. Intuitively, an extremal realisation is a most economic causal history associated with its image, a configuration of \mathcal{A} ; it is extremal in being a realisation with minimal causal dependencies.

Any realisation in \mathcal{A} can be coarsened to an extremal realisation.

► **Lemma 3.** *For any realisation ρ there is an extremal realisation ρ' with $\rho \succeq_2^f \rho'$.*

Proof. The category of realisations with total maps has colimits of total-order diagrams. A diagram d from a total order (I, \leq) to realisations, comprises a collection of total maps of realisations $d_{i,j} : d(i) \rightarrow d(j)$ when $i \leq j$ s.t. $d_{i,i}$ is always the identity map and if $i \leq j$ and $j \leq k$ then $d_{i,k} = d_{j,k} \circ d_{i,j}$. We suppose each realisation $d(i)$ has carrier (E_i, \leq_i) with $d(i) : E_i \rightarrow A$. We construct the colimit realisation of the diagram as follows.

The elements of the colimit realisation consist of equivalence classes of elements of the disjoint union $E =_{\text{def}} \bigsqcup_{i \in I} E_i$ under the equivalence

$$(i, e_i) \sim (j, e_j) \iff \exists k \in I. i \leq k \ \& \ j \leq k \ \& \ d_{i,k}(e_i) = d_{j,k}(e_j).$$

Consequently we may define a function $\rho_E : E \rightarrow A$ by taking $\rho_E(\{e_i\}_{\sim}) = \rho_i(e_i)$. Because every $d_{i,j}$ is a surjective function, every equivalence class in E has a representative in E_i for every $i \in I$. Moreover, for any $e \in E$ there is $k \in I$ s.t.

$$\{e' \in E \mid e' \leq_E e\} = \{\{e'_k\}_{\sim} \mid e'_k \leq_k e_k\},$$

where $e = \{e_k\}_{\sim}$, so is finite. It follows that ρ_E is a realisation. The maps $f_i : \rho_i \succeq_2 \rho_E$, where $i \in I$, given by $f_i(e_i) = \{e_i\}_{\sim}$ form a colimiting cocone.

Suppose ρ is a realisation. Consider all total-order diagrams d from a total order (I, \leq) to realisations starting from ρ with $d_{i,j}$ not an isomorphism if $i < j$. Amongst them, by Zorn's lemma, there is a maximal diagram w.r.t. extension. From the maximality of the diagram its colimit is necessarily extremal. In more detail, construct a colimiting cocone $f_i : d(i) \succeq_2 \rho_E$, $i \in I$, with the same notation as above. By maximality of the diagram some f_k must be an isomorphism; otherwise we could extend the diagram by adding a top element to the total order and sending it to ρ_E . If j should satisfy $k < j$ then $f_j \circ d_{k,j} = f_k$ so $f_k^{-1} \circ f_j \circ d_{k,j} = \text{id}_{E_k}$. It would follow that $d_{k,j}$ is injective, as well as surjective, it being a total map of realisations, and consequently that $d_{k,j}$ is an isomorphism – a contradiction. Hence k is the maximum element in (I, \leq) . If the colimit were not extremal we could again adjoin a new top element above k thus extending the diagram – a contradiction. ◀

For example, as a corollary, a countable configuration of a family of configurations always has an injective extremal realisation. By serialising the countable configuration, $a_1 \leq a_2 \leq \dots \leq a_n \leq \dots$, where $\{a_1, \dots, a_n\} \in \mathcal{A}$ for all n , we obtain an injective realisation ρ . By Lemma 3 we can coarsen ρ to an extremal realisation ρ' with $\rho \succeq_2^f \rho'$. As $\rho = \rho' f$ the surjective function f is also injective, so a bijection, ensuring that the extremal realisation ρ' is injective.

The following rather technical lemma and corollary are crucial.

► **Lemma 4.** *Assume $(R, \leq), \rho, (R_0, \leq_0), \rho_0$ and $(R_1, \leq_1), \rho_1$ are realisations.*

(i) Suppose $f = \rho \succeq_{-1}^{f_1} \rho_0 \succeq_{-2}^{f_2} \rho_1$. Then there are maps so that $f = \rho \succeq_{-2}^{g_2} \rho' \succeq_{-1}^{g_1} \rho_1$:

$$\begin{array}{ccc} \rho & \xrightarrow{g_2} & \rho' \\ f_1 \downarrow & & \downarrow g_1 \\ \rho_0 & \xrightarrow{f_2} & \rho_1 \end{array}$$

(ii) Suppose $\rho \succeq_{-1}^{f_1} \rho_0$ where R_0 is not a down-closed subset of R . Then there are maps so that $f_1 = \rho \succeq_{-2}^{g_2} \rho' \succeq_{-1}^{g_1} \rho_0$ with g_2 not an isomorphism:

$$\begin{array}{ccc} \rho & \xrightarrow{g_2} & \rho' \\ f_1 \downarrow & \nearrow g_1 & \\ \rho_0 & & \end{array}$$

Proof.

(i) Construct the realisation $(R', \leq'), \rho'$ as follows. Define

$$R' = (R \setminus R_0) \cup R_1$$

where w.l.o.g. we assume the sets $R \setminus R_0$ and R_1 are disjoint. Define $g_2 : R \rightarrow R'$ to act as the identity on elements of $R \setminus R_0$ and as f_2 on elements of R_0 .

When $b \in R \setminus R_0$, define

$$a \leq' b \text{ iff } \exists a_0 \in R. a_0 \leq b \ \& \ g_2(a_0) = a.$$

When $b \in R_1$, define

$$a \leq' b \text{ iff } a \in R_1 \ \& \ a \leq_1 b.$$

To see \leq' is a partial order observe that reflexivity and antisymmetry follow directly from the corresponding properties of \leq and \leq_1 . Transitivity requires an argument by cases. For example, in the most involved case, where

$$c \leq' a \text{ with } a \in R_1 \text{ and } a \leq' b \text{ with } b \in R \setminus R_0$$

we obtain

$$c \leq_1 a \text{ and } a_0 \leq b$$

for some $a_0 \in R_0$ with $f_2(a_0) = a$. As f_2 is surjective and preserves down-closed subsets,

$$c_0 \leq_0 a_0 \text{ and } a_0 \leq b$$

for some $c_0 \in R_0$ with $f_2(c_0) = c$. Consequently, $c_0 \leq b$ with $g_2(c_0) = c$, making $c \leq' b$, as required for transitivity.

Define ρ' to act as ρ on elements of $R \setminus R_0$ and as ρ_1 on elements of R_1 . Then $\rho = \rho' g_2$ directly. We check ρ' preserves down-closed subsets, so is a realisation. Let $b \in R'$. If $b \in R_1$ then $\rho'[b]' = \rho_1[b]_1 \in \mathcal{A}$. If $b \in R \setminus R_0$ then $\rho'[b]' = \rho g_2[b]$ is the image under ρ of the down-closed subset $g_2[b]$, so in \mathcal{A} . Because f_2 preserves down-closed subsets so does g_2 . We already have $\rho = \rho' g_2$, making g_2 a map of realisations $\rho \succeq_{-2}^{g_2} \rho'$. Define $g_1 : R' \rightarrow R_1$ to be the reverse of the inclusion $R_1 \subseteq R'$. Because ρ_1 is the restriction of ρ' to R_1 , g_1 is a map of realisations $\rho' \succeq_{-1}^{g_1} \rho_1$. By construction $f = g_1 g_2$.

(ii) This follows from the construction of $(R' \leq')$, ρ' used in (i) but in the special case where f_2 is the identity map (with $R_0 = R_1$). Then $R' = R$ but $\leq' \neq \leq$ as there is $e \in R_0$ with $[e]_0 \subsetneq [e]$ ensuring that $[e]' = [e]_0 \neq [e]$. ◀

► **Corollary 5.** *If ρ is extremal and $\rho \succeq^f \rho'$, then ρ' is extremal and there is ρ_0 s.t. $f : \rho \succeq_1 \rho_0 \cong \rho'$. Moreover, the carrier R_0 of ρ_0 is a down-closed subset of the carrier R of ρ , with order the restriction of that on R .*

Proof. Directly from Lemma 4. Assume ρ is extremal and $\rho \succeq^f \rho'$. We can factor f into $\rho \succeq_1^{f_1} \rho_0 \succeq_2^{f_2} \rho'$. From (i), if ρ_0 were not extremal nor would ρ be – a contradiction; hence f_2 is an isomorphism. From (ii), the carrier R_0 of ρ_0 has to be a down-closed subset of the carrier R of ρ , as otherwise we would contradict the extremality of ρ . ◀

It follows that if ρ is extremal and $\rho \succeq^f \rho'$ then ρ' is extremal and the inverse relation $g =_{\text{def}} f^{-1}$ is an injective function preserving and reflecting down-closed subsets, *i.e.* $g[r'] = [g(r')]$ for all $r' \in R'$. In other words:

► **Corollary 6.** *If ρ is extremal and $\rho \succeq^f \rho'$, then ρ' is extremal and the inverse $g =_{\text{def}} f^{-1}$ is a rigid embedding from the carrier of ρ' to the carrier of ρ such that $\rho' = \rho g$.*

► **Lemma 7.** *Let $(R, \leq), \rho$ be an extremal realisation. Then*

- (i) *if $r' \leq r$ and $\rho(r) = \rho(r')$ then $r = r'$;*
- (ii) *if $[r] = [r']$ and $\rho(r) = \rho(r')$ then $r = r'$. Here $[r] =_{\text{def}} [r] \setminus \{r\}$.*

Proof.

(i) Suppose $r' \leq r$ and $\rho(r) = \rho(r')$. By Corollary 6, we may project to $[r]$ to obtain an extremal realisation $\rho_0 : [r] \rightarrow A$. Suppose r and r' were unequal. We can define a realisation as the restriction of ρ_0 to $[r]$. The function from $[r]$ to $[r]$ taking r to r' and otherwise acting as the identity function is a map of realisations from the realisation ρ_0 and clearly not an isomorphism, showing ρ_0 to be non-extremal – a contradiction. Hence $r = r'$, as required.

(ii) Suppose $[r] = [r']$ and $\rho(r) = \rho(r')$. Projecting to $[\{r, r'\}]$ we obtain an extremal realisation. If r and r' were unequal there would be a non-isomorphism map to the realisation obtained by projecting to $[r]$, *viz.* the map from $[\{r, r'\}]$ to $[r]$ sending r' to r and fixing all other elements. ◀

In fact, by modifying condition (i) in the lemma above a little we can obtain a characterisation of extremal realisations – though not strictly necessary for the rest of the paper:

► **Lemma 8.** *Let $(R, \leq), \rho$ be a realisation. Then ρ is extremal iff*

- (i) *if $X \subseteq [r]$, with X down-closed and $r \in R$, and $\rho(X \cup \{r\}) \in \mathcal{A}$ then $X = [r]$; and*
- (ii) *if $[r] = [r']$ and $\rho(r) = \rho(r')$ then $r = r'$.*

Proof. “Only if”: Assume ρ is extremal. We have already established (ii) in Lemma 7. To show (i), suppose X is down-closed and $X \subseteq [r]$ in R with $\rho(X \cup \{r\}) \in \mathcal{A}$. By Corollary 6, we may project to $[r]$ to obtain an extremal realisation $\rho_0 : [r] \rightarrow A$. Modify the restricted order $[r]$ to one in which $r' \leq r$ iff $r' \in X \cup \{r\}$, and is otherwise unchanged. The same underlying function ρ_0 remains a realisation, call it ρ'_0 , on the modified order. The identity function gives us a map $f : \rho_0 \succeq_2 \rho'_0$ which is an isomorphism between realisations iff $X = [r]$. “If”: Assume (i) and (ii). Suppose $f : \rho \succeq_2 \rho'$, where R', ρ' is a realisation. We show f is

injective and order-preserving. As f is presumed to be surjective and to preserve down-closed subsets we can then conclude it is an isomorphism.

To see f is injective suppose the contrary that $f(r_1) = f(r_2)$ for $r_1 \neq r_2$. W.l.o.g. we may suppose r_1 and r_2 are minimal in the sense that

$$r'_1 \neq r'_2 \ \& \ r'_1 \leq r_1 \ \& \ r'_2 \leq r_2 \ \& \ f(r'_1) = f(r'_2) \implies r'_1 = r_1 \ \& \ r'_2 = r_2.$$

Define $r' =_{\text{def}} f(r_1) = f(r_2)$. Then

$$[r'] \subseteq f[r_1] \ \& \ [r'] \subseteq f[r_2].$$

Furthermore, as only r' can be the image of r_1 and r_2 under the function f ,

$$[r'] \subseteq f[r_1] \ \& \ [r'] \subseteq f[r_2].$$

It follows that

$$[r'] \subseteq f[r_1] \cap f[r_2] = f([r_1] \cap [r_2])$$

where the equality is a consequence of the minimality of r_1, r_2 . Taking $X =_{\text{def}} [r_1] \cap [r_2]$ we have $(fX) \cup \{r'\}$ is down-closed in R' . Therefore

$$\rho(X \cup \{r_1\}) = \rho'f(X \cup \{r_1\}) = \rho'(fX \cup \{r'\}) \in \mathcal{A}.$$

By condition (i), $X = [r_1]$. Similarly, $X = [r_2]$, so $[r_1] = [r_2]$. Obviously $\rho(r_1) = \rho'f(r_1) = \rho'f(r_2) = \rho(r_2)$, so we obtain $r_1 = r_2$ by (ii) – a contradiction, so f is injective.

We now check that f preserves the order. Let $r \in R$. Define

$$X =_{\text{def}} [\{r_1 \leq r \mid f(r_1) < f(r)\}],$$

where the square brackets signify down-closure in R . Then X is down-closed in R by definition and $X \subseteq [r]$. We have $[f(r)] \subseteq f[r]$ whence

$$fX = f[r] \cap [f(r)] = [f(r)].$$

Therefore $fX \cup \{f(r)\}$ is down-closed in R' , so

$$\rho(X \cup \{r\}) = \rho'f(X \cup \{r\}) = \rho'(fX \cup \{f(r)\}) \in \mathcal{A}.$$

Hence $X = [r]$, by (i). It follows that

$$r_1 < r \implies r_1 \in X \implies f(r_1) < f(r) \text{ in } R'.$$

This shows that f preserves the order on R . ◀

► **Lemma 9.** *There is at most one map between extremal realisations.*

Proof. Let $(R, \leq), \rho$ and $(R', \leq'), \rho'$ be extremal realisations. Let $f, f' : \rho \rightarrow \rho'$ be maps with converse relations g and g' respectively. We show the two functions g and g' are equal, and hence so are their converses f and f' . Suppose otherwise that $g \neq g'$. Then there is an \leq -minimal $r' \in R'$ for which $g(r') \neq g'(r')$ and $g[r'] = g'[r']$. Hence $[g(r')] = [g'(r')]$ and $\rho(g(r')) = \rho'(r') = \rho(g'(r'))$. As ρ is extremal, by Lemma 7(ii) we obtain $g(r') = g'(r')$ – a contradiction. ◀

9:12 Causal Unfoldings

Hence extremal realisations of A under \preceq form a preorder. The *order of extremal realisations* has as elements isomorphism classes of extremal realisations ordered according to the existence of a map between representatives of isomorphism classes. Alternatively, we could take a choice of representative from each isomorphism class and order these according to whether there is a map from one to the other. Recall a prime extremal realisation is an extremal realisation with a top element, *i.e.* when its carrier contains an element which dominates all other elements in the carrier. The following is a direct corollary of Proposition 14 in the next section.

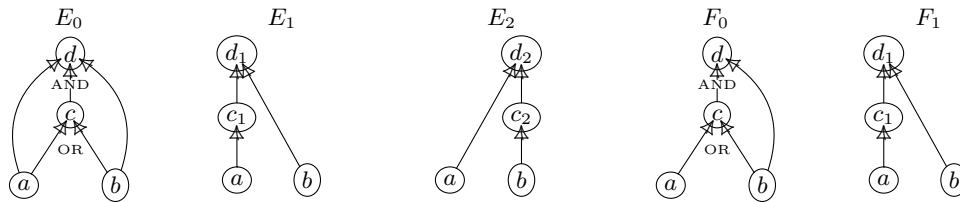
► **Proposition 10.** *The order of extremal realisations of a family of configurations A forms a prime-algebraic domain [10] with complete primes the prime extremal realisations.*

The proofs of the following observations are straightforward consequences of the definitions. They emphasise that prime extremal realisations are a generalisation of complete primes.

► **Proposition 11.** *Let (A, \leq_A, Con_A) be a prime event structure. For an extremal realisation $(R, \leq_R), \rho$ of $\mathcal{C}^\infty(A)$, the function $\rho : R \rightarrow \rho R$ is an order isomorphism between (R, \leq_R) and the configuration $\rho R \in \mathcal{C}^\infty(A)$ ordered by the restriction of \leq_A . The function taking an extremal realisation $(R, \leq_R), \rho$ to the configuration ρR is an order isomorphism from the order of extremal realisations of $\mathcal{C}^\infty(A)$ to the configurations of A ; prime extremal realisations correspond to complete primes of $\mathcal{C}^\infty(A)$.*

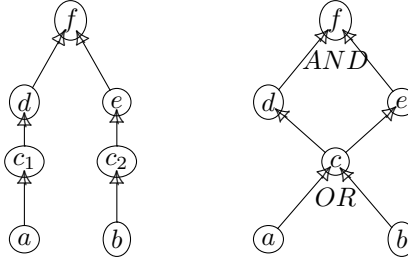
A configuration $x \in \mathcal{F}$, of a family of configurations \mathcal{F} , is *irreducible* iff there is a necessarily unique $e \in x$ such that $\forall y \in \mathcal{F}, e \in y \subseteq x$ implies $y = x$. Irreducibles coincide with complete (join) irreducibles w.r.t. the order of inclusion. It is tempting to think of irreducibles as representing minimal complete enablings. But, as sets, irreducibles both (1) lack sufficient structure: in the formulation we are led to, of minimal complete enablings as prime extremal realisations, several prime realisations can have the same irreducible as their underlying set; and (2) are not general enough: there are prime realisations whose underlying set is not an irreducible. We conclude with examples illustrating the nature of extremal realisations; it is convenient to describe families of configurations by general event structures.

► **Example 12.** This example shows that prime extremal realisations do not correspond to irreducible configurations. First, we show a general event structure E_0 (all subsets consistent) with irreducible configuration $\{a, b, c, d\}$ and two (injective) prime extremals E_1 and E_2 with tops d_1 and d_2 which both have the same irreducible configuration $\{a, b, c, d\}$ as their image. The lettering indicates the functions associated with the realisations, *e.g.* events d_1 and d_2 in the partial orders map to d in the general event structure.



On the other hand there are prime extremal realisations of which the image is not an irreducible configuration. Consider the general event structure F_0 . The prime extremal F_1 describes a situation where d is enabled by b and c , and c is enabled by a . It has image the configuration $\{a, b, c, d\}$ which is not irreducible, being the union of the two incomparable configurations $\{a\}$ and $\{b, c, d\}$.

► **Example 13.** It is possible to have extremal realisations in which an event depends on an event of the family having been enabled in two distinct ways, as in the following prime extremal realisation, on the left; it is clearly not injective.



The extremal describes the event f being enabled by d and e where they are in turn enabled by different ways of enabling c . We assume all subsets consistent.

5 The causal unfolding: an adjunction from \mathcal{E}_{\equiv} to \mathcal{Fam}_{\equiv}

Furnished with the concept of extremal realisation, we can now exhibit an adjunction (precisely, a very simple case of biadjunction or pseudo adjunction) from \mathcal{E}_{\equiv} , the category of ese's, to \mathcal{Fam}_{\equiv} , the category of equivalence families. The left adjoint $I : \mathcal{E}_{\equiv} \rightarrow \mathcal{Fam}_{\equiv}$ is the full and faithful functor which takes an ese to its family of configurations with the original equivalence.

The right adjoint, the *causal unfolding*, $er : \mathcal{Fam}_{\equiv} \rightarrow \mathcal{E}_{\equiv}$ is defined on objects as follows. Let \mathcal{A} be an equivalence family with underlying set A . Define $er(\mathcal{A}) = (P, \text{Con}_P, \leq_P, \equiv_P)$ where

- P consists of a choice from within each isomorphism class of the prime extremals p of \mathcal{A} – we write $top(p)$ for the image of the top element in A ;
- Causal dependency \leq_P is \preceq on P ;
- $X \in \text{Con}_P$ iff $X \subseteq_{\text{fin}} P$ and $top[X]_P \in \mathcal{A}$ – the set $[X]_P$ is the \leq_P -downwards closure of X , so equal to $\{p' \in P \mid \exists p \in X. p' \preceq p\}$;
- $p_1 \equiv_P p_2$ iff $p_1, p_2 \in P$ and $top(p_1) \equiv_A top(p_2)$.

► **Proposition 14.** *The configurations of P , ordered by inclusion, are order-isomorphic to the order of extremal realisations: an extremal realisation ρ corresponds, up to isomorphism, to the configuration $\{p \in P \mid p \preceq \rho\}$ of P ; conversely, a configuration x of P corresponds to an extremal realisation $top : x \rightarrow A$ with carrier (x, \preceq) , the restriction of the order of P to x .*

Proof. It will be helpful to recall, from Corollary 6, that if $\rho \succeq^f \rho'$ between extremal realisations, then the inverse relation f^{-1} is a rigid embedding of (the carrier of) ρ' in (the carrier of) ρ ; so $\rho' \preceq \rho$ stands for a rigid embedding. Suppose $x \in \mathcal{C}^\infty(P)$. Then x determines an extremal realisation

$$\theta(x) =_{\text{def}} top : (x, \preceq) \rightarrow A.$$

The function $\theta(x)$ is a realisation because each p in x is, and extremal because, if not, one of the p in x would fail to be extremal, a contradiction. Clearly $\rho' \preceq \rho$ implies $\theta(\rho') \subseteq \theta(\rho)$. Conversely, it is easily checked that any extremal realisation $\rho : (R, \leq) \rightarrow A$ defines a configuration $\{p \in P \mid p \preceq \rho\}$. If $x \subseteq y$ in $\mathcal{C}^\infty(P)$ then $\theta(x) \preceq \theta(y)$. It can be checked that θ and ϕ are mutual inverses, i.e. $\phi\theta(x) = x$ and $\theta\phi(\rho) \cong \rho$ for all configurations x of P and extremal realisations ρ . ◀

From the above proposition we see that the events of $er(\mathcal{A})$ correspond to the order-theoretic completely-prime extremal realisations [10]. This justifies our use of the term “prime extremal” for extremal with top element.

The component of the counit of the adjunction $\epsilon_A : I(er(\mathcal{A})) \rightarrow \mathcal{A}$ is given by the function

$$\epsilon_A(p) = top(p).$$

It is a routine check to see that ϵ_A preserves \equiv and that any configuration x of P images under top to a configuration in \mathcal{A} , moreover in a way that reflects \equiv .

► **Theorem 15.** *Let $\mathcal{A} \in \mathcal{Fam}_{\equiv}$. For all $f : I(Q) \rightarrow \mathcal{A}$ in \mathcal{Fam}_{\equiv} , there is a map $h : Q \rightarrow er(\mathcal{A})$ in \mathcal{E}_{\equiv} such that $f = \epsilon_A \circ I(h)$, i.e. so the diagram*

$$\begin{array}{ccc} A & \xleftarrow{\epsilon_A} & I(er(\mathcal{A})) \\ & \searrow f & \uparrow I(h) \\ & & I(Q) \end{array}$$

commutes. Moreover, if $h' : Q \rightarrow er(\mathcal{A})$ is a map in \mathcal{E}_{\equiv} s.t. $f \equiv \epsilon_A \circ I(h')$, i.e. the diagram above commutes up to \equiv , then $h' \equiv h$.

Proof. Let $Q = (Q, \text{Con}_Q, \leq_Q, \equiv_Q)$ be an ese and $f : I(Q) \rightarrow \mathcal{A}$ a map in \mathcal{Fam}_{\equiv} . We shall define a map $h : Q \rightarrow er(\mathcal{A})$ s.t. $f = \epsilon_A h$. (As here, in the proof we shall elide the composition symbol \circ , and I on maps which it leaves unchanged.)

We define the map $h : Q \rightarrow er(\mathcal{A})$ by induction on the depth of Q . The depth of an event in an event structure is the length of a longest \leq -chain up to it – so an initial event has depth 1. We take the depth of an event structure to be the maximum depth of its events. (Because of our reliance on Lemma 3, we use the axiom of choice implicitly.)

Assume inductively that $h^{(n)}$ defines a map from $Q^{(n)}$ to $er(\mathcal{A})$ where $Q^{(n)}$ is the restriction of Q to depth below or equal to n such that $f^{(n)}$ the restriction of f to $Q^{(n)}$ satisfies $f^{(n)} = \epsilon_A h^{(n)}$. (In particular, $Q^{(0)}$ is the empty ese and $h^{(0)}$ the empty function.) Then, by Proposition 14, any configuration x of $Q^{(n)}$ determines an extremal realisation $\rho_x : h^{(n)}x \rightarrow A$ with carrier $(h^{(n)}x, \preceq)$.

Suppose $q \in Q$ has depth $n + 1$. If $f(q)$ is undefined take $h^{(n+1)}(q)$ to be undefined. Otherwise, note there is an extremal realisation $\rho_{[q]}$ with carrier $(h[q], \preceq)$. Extend $\rho_{[q]}$ to a realisation $\rho_{[q]}^\top$ with carrier that of $\rho_{[q]}$ with a new top element \top adjoined, and make $\rho_{[q]}^\top$ extend the function $\rho_{[q]}$ by taking \top to $f(q)$. By Lemma 3, there is an extremal realisation ρ such that $\rho_{[q]}^\top \succeq_2 \rho$. Because $\rho_{[q]}$ is extremal, $\rho \succeq_1 \rho_{[q]}$, so ρ only extends the order of $\rho_{[q]}$ with extra dependencies of \top . (For notational simplicity we identify the carrier of ρ with the set $h[q] \cup \{\top\}$.) Project ρ to the extremal with top \top . Define this to be the value of $h^{(n+1)}(q)$. In this way, we extend $h^{(n)}$ to a partial function $h^{(n+1)} : Q^{(n+1)} \rightarrow er(\mathcal{A})$ such that $f^{(n+1)} = \epsilon_A h^{(n+1)}$. To see that $h^{(n+1)}$ is a map we can use Proposition 2. By construction $h^{(n+1)}$ satisfies property (ii) of Proposition 2 and the other properties are inherited fairly directly from f via the definition of $er(\mathcal{A})$.

Defining $h = \bigcup_{n \in \omega} h^{(n)}$ we obtain a map $h : Q \rightarrow er(\mathcal{A})$ such that $f = \epsilon_A h$.

Suppose $h' : Q \rightarrow er(\mathcal{A})$ is a map s.t. $f \equiv \epsilon_A h'$. Then, for any $q \in Q$,

$$top(h'(q)) = \epsilon_A h'(q) \equiv_A f(q) = \epsilon_A h(q) = top(h(q)),$$

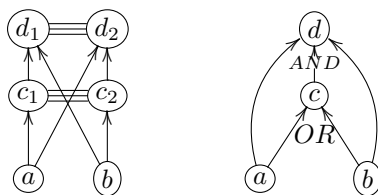
so $h'(q) \equiv_P h(q)$ in $er(\mathcal{A})$. Thus $h' \equiv h$. ◀

The theorem does not quite exhibit a traditional adjunction, because the usual cofreeness condition specifying an adjunction is weakened to only having uniqueness up to \equiv . However the condition it describes does specify an exceedingly simple case of a pseudo adjunction (or biadjunction) between 2-categories – a set together with an equivalence relation (a *setoid*) is a very simple example of a category. As a consequence, whereas the usual cofreeness condition allows us to extend the right adjoint to arrows, so obtaining a functor, in this case following that same line will only yield a pseudo functor er as right adjoint: thus extended, er will only preserve composition and identities up to \equiv .

The map $(P, \equiv) \rightarrow er(\mathcal{C}^\infty(P), \equiv)$ which takes $p \in P$ to the realisation with carrier $([p], \leq)$, the restriction of the causal dependency of P , with the inclusion function $[p] \hookrightarrow P$ is an isomorphism; recall from Proposition 11 that the configurations of a prime event structure correspond to its extremal realisations. Such maps furnish the components of the unit of the pseudo adjunction:

$$\mathcal{E}_\equiv \begin{array}{c} \xleftarrow{er} \\ \top \\ \xrightarrow{I} \end{array} \mathcal{Fam}_\equiv$$

► **Example 16.** On the right we show a general event structure (all subsets consistent) and on its left its causal unfolding to an ese under er ; the unfolding's events are the prime extremals.⁶



6 Unfolding general event structures

Recall \mathcal{G} is the category of general event structures. We obtain a pseudo adjunction from \mathcal{E}_\equiv to \mathcal{G} via an adjunction from \mathcal{Fam}_\equiv to \mathcal{G} . The right adjoint $fam : \mathcal{G} \rightarrow \mathcal{Fam}_\equiv$ is most simply described. Given (E, Con, \vdash) in \mathcal{G} it returns the equivalence family $(\mathcal{C}^\infty(E), \equiv)$ in \mathcal{Fam}_\equiv comprising the configurations together with the identity equivalence between events that appear within some configuration; the partial functions between events that are maps in \mathcal{G} are automatically maps in \mathcal{Fam}_\equiv – the action of fam on maps.

For the effect of the left adjoint $col : \mathcal{Fam}_\equiv \rightarrow \mathcal{G}$ on objects, define the *collapse*

$$col(\mathcal{A}) =_{\text{def}} (E, \text{Con}, \vdash)$$

where

- $E = A_\equiv$, the equivalence classes of events in $A =_{\text{def}} \bigcup \mathcal{A}$;
- $X \in \text{Con}$ iff $X \subseteq_{\text{fin}} y_\equiv =_{\text{def}} \{\{a\}_\equiv \mid a \in y\}$, for some $y \in \mathcal{A}$; and
- $X \vdash e$ iff $e \in E$, $X \in \text{Con}$ and $e \in y_\equiv \subseteq X \cup \{e\}$, for some $y \in \mathcal{A}$.

It follows that y_\equiv is a configuration of $col(\mathcal{A})$ whenever $y \in \mathcal{A}$. From this it is easy to see that $col(\mathcal{A})$ is a replete general event structure.

Let $(\mathcal{A}, \equiv) \in \mathcal{Fam}_\equiv$. Assume that \mathcal{A} has underlying set A . The unit of the adjunction is defined to have typical component $\eta_A : (\mathcal{A}, \equiv) \rightarrow fam(col(\mathcal{A}, \equiv))$ given by $\eta_A(a) = \{a\}_\equiv$. It is easy to check that η_A is a map in \mathcal{Fam}_\equiv .

⁶ See [6] for further examples of the causal unfolding including an inductive characterisation in 5.2.2.

9:16 Causal Unfoldings

► **Theorem 17.** *Suppose that $B = (B, \text{Con}_B, \vdash_B) \in \mathcal{G}$ and that $g : (\mathcal{A}, \equiv) \rightarrow (\mathcal{C}^\infty(B), =)$ is a map in \mathcal{Fam}_\equiv . Then, there is a unique map $k : \text{col}(\mathcal{A}, \equiv) \rightarrow B$ in \mathcal{G} such that the diagram*

$$\begin{array}{ccc} (\mathcal{A}, \equiv) & \xrightarrow{\eta^{\mathcal{A}}} & \text{fam}(\text{col}(\mathcal{A}, \equiv)) \\ & \searrow g & \downarrow \text{fam}(k) \\ & & (\mathcal{C}^\infty(B), =) \end{array}$$

commutes.

Proof. The map $k : \text{col}(\mathcal{A}, \equiv) \rightarrow B$ is given as the function $k(e) = g(a)$ where $e = \{a\}_\equiv$. It is easily checked to be a map in \mathcal{G} and moreover to be the unique map from $\text{col}(\mathcal{A}, \equiv)$ to B making the above diagram commute. ◀

Theorem 17 determines an adjunction:

$$\mathcal{Fam}_\equiv \begin{array}{c} \xleftarrow{\text{fam}} \\ \top \\ \xrightarrow{\text{col}} \end{array} \mathcal{G}$$

The construction col automatically extends from objects to maps; maps in \mathcal{Fam}_\equiv preserve equivalence so collapse to functions preserving equivalence classes. The counit of the adjunction has components $\epsilon_E : \text{col}((\mathcal{C}^\infty(E), =)) \rightarrow E$ which send singleton equivalence classes $\{e\}$ to e . The counit is an isomorphism at precisely those general event structures E which are replete, so cuts down to a reflection from the subcategory of replete general event structures into equivalence families.

Composing

$$\mathcal{E}_\equiv \begin{array}{c} \xleftarrow{\text{er}} \\ \top \\ \xrightarrow{I} \end{array} \mathcal{Fam}_\equiv \begin{array}{c} \xleftarrow{\text{fam}} \\ \top \\ \xrightarrow{\text{col}} \end{array} \mathcal{G}$$

we obtain a pseudo adjunction

$$\mathcal{E}_\equiv \begin{array}{c} \xleftarrow{\quad} \\ \top \\ \xrightarrow{\quad} \end{array} \mathcal{G}.$$

Its right adjoint constructs the *causal unfolding* of a general event structure.

The composite pseudo adjunction from \mathcal{E}_\equiv to \mathcal{G} cuts down to a reflection, in the sense that the counit is a natural isomorphism, when we restrict to the subcategory of \mathcal{G} where all general event structures are replete. Then the right adjoint provides a pseudo functor embedding replete general event structures (and so families of configurations) in ese's.⁷

⁷ We deal with a possible source of confusion. There is an obvious “inclusion” functor from the category of prime event structures \mathcal{E} to the category \mathcal{E}_\equiv : it extends an event structure with the identity equivalence. Regarding \mathcal{E}_\equiv as a plain category, so dropping the enrichment by equivalence relations, the “inclusion” functor $\mathcal{E} \hookrightarrow \mathcal{E}_\equiv$ has a right adjoint, *viz.* the forgetful functor which given an ese (E, \equiv) produces the event structure E by dropping the equivalence. The adjunction is a coreflection because the inclusion functor is full. Might not this coreflection compose with the pseudo adjunction from \mathcal{E}_\equiv to \mathcal{G} to produce a pseudo adjunction from \mathcal{E} to \mathcal{G} ? No. Clearly the coreflection is not enriched in equivalence relations; the natural bijection of the coreflection cannot respect the equivalence on maps. For this reason the two different forms of adjunction do not compose to yield a pseudo adjunction from \mathcal{E} to \mathcal{G} .

7 Conclusion

This concludes the construction of causal unfoldings of (very general) equivalence-families, and so, in particular, general event structures. In applications it has been useful to cut down the unfolding to subcategories. In particular, while the category of event structures with equivalence, \mathcal{E}_{\equiv} , does have bipullbacks (in which commutations and uniqueness are only up to the equivalence \equiv on maps) it doesn't always have the pseudo pullbacks or pullbacks, used in defining the composition of strategies [2, 1]. However, an important subcategory does: define \mathcal{EDC} to be the subcategory of \mathcal{E}_{\equiv} with objects, *event structures with disjunctive causes* (edc's), satisfying

$$p_1, p_2 \leq p \ \& \ p_1 \equiv p_2 \implies p_1 = p_2.$$

In an edc an event cannot causally depend on two distinct prime causes of a common disjunctive event, and so rules out realisations such as that mentioned in Example 13. \mathcal{EDC} provides a suitable foundation for probabilistic strategies with parallel causes and is handily related by adjunctions to general and prime event structures [7].

References

- 1 Simon Castellan, Pierre Clairambault, and Glynn Winskel. 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 '14, Vienna, Austria, July 14 - 18, 2014*. ACM, 2014.
- 2 Pierre Clairambault, Julian Gutierrez, and Glynn Winskel. The Winning Ways of Concurrent Games. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*. IEEE Computer Society, 2012.
- 3 Ioana Cristescu. *Operational and denotational semantics for the reversible pi-calculus*. PhD thesis, PPS, Université Paris Diderot, 2015.
- 4 Ioana Cristescu, Jean Krivine, and Daniele Varacca. Rigid families for CCS and the pi-Calculus. In *International Colloquium on Theoretical Aspects of Computing ICTAC, 12th ed. Cali, Colombia, 2015*.
- 5 Vincent Danos, Jerome Feret, Walter Fontana, Russell Harmer, Jonathan Hayman, Jean Krivine, Chris Thompson-Walsh, and Glynn Winskel. Graphs, Rewriting and Pathway Reconstruction for Rule-Based Models. In *FSTTCS 2012*, volume 18 of *LIPICs*, pages 276–288. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012.
- 6 Marc de Visme. Cambridge Internship Report, ENS Paris. Available from Glynn Winskel's homepage <http://www.cl.cam.ac.uk/~gw104/mdv-report.pdf>, 2015.
- 7 Marc de Visme and Glynn Winskel. Strategies with Parallel Causes. In *CSL 2017*, volume 82 of *LIPICs*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017.
- 8 G. M. Kelly. *Basic concepts of enriched category theory*. LNM 64. CUP, 1982.
- 9 Y. Kinoshita and J. Power. Category theoretic structure of setoids. *TCS*, 546, 2014.
- 10 Mogens Nielsen, Gordon Plotkin, and Glynn Winskel. Petri Nets, Event Structures and Domains. *TCS*, 13:85–108, 1981.
- 11 Judea Pearl. *Causality*. CUP, 2013.
- 12 John Power. 2-Categories. *BRICS Notes Series NS-98-7*, 1998.
- 13 Glynn Winskel. *Events in computation*. Edinburgh University, 1980. PhD thesis, Edinburgh.
- 14 Glynn Winskel. Event Structures. In *Advances in Petri Nets*, LNCS 255, 1986.
- 15 Glynn Winskel and Mogens Nielsen. Models for concurrency. In Samson Abramsky and Dov Gabbay, editors, *Semantics and Logics of Computation*. OUP, 1995.

A Equiv-enriched categories

Here we explain in more detail what we mean when we say “enriched in the category of sets with equivalence relations” and employ terms such as “enriched adjunction,” “pseudo adjunction” and “pseudo pullback.” The classic text on enriched categories is [8], but for this paper the articles [9] and [12] provide short, accessible introductions to the notions we use from Equiv-enriched categories and 2-categories, respectively.

Equiv is the category of *equivalence relations*. Its objects are (A, \equiv_A) comprising a set A and an equivalence relation \equiv_A on it. Its maps $f : (A, \equiv_A) \rightarrow (B, \equiv_B)$ are total functions $f : A \rightarrow B$ which preserve equivalence.

We shall use some basic notions from enriched category theory [8]. We shall be concerned with categories enriched in Equiv, called Equiv-enriched categories, in which the homsets possess the structure of equivalence relations, respected by composition [9]. This is the sense in which we say categories are enriched in (the category of) equivalence relations. We similarly borrow the concept of an Equiv-enriched functor between Equiv-enriched categories for a functor which preserves equivalence in acting on homsets. An Equiv-enriched adjunction is a usual adjunction in which the natural bijection of the adjunction preserves and reflects equivalence.

Because an object in Equiv can be regarded as a (very simple) category, we can regard Equiv-enriched categories as (very simple) 2-categories to which notions from 2-categories apply [12].

A *pseudo functor* between Equiv-enriched categories is like a functor but the usual laws only need hold up to equivalence. A *pseudo adjunction* (or biadjunction) between 2-categories permits a weakening of the usual natural isomorphism between homsets, now also categories, to a natural equivalence of categories. In the special case of a pseudo adjunction between Equiv-enriched categories the equivalence of homset categories amounts to a pair of \equiv -preserving functions whose compositions are \equiv -equivalent to the identity function. With traditional adjunctions, by specifying the action of one adjoint solely on objects, we determine it as a functor; with pseudo adjunctions we can only determine it as a pseudo functor – in general a pseudo adjunction relates two pseudo functors. Pseudo adjunctions compose in the expected way. An Equiv-enriched adjunction is a special case of a 2-adjunction between 2-categories and a very special case of pseudo adjunction. In Section 6 we compose an Equiv-enriched adjunction with a pseudo adjunction to obtain a new pseudo adjunction.

Similarly we can specialise the notions pseudo pullbacks and bipullbacks from 2-categories to Equiv-enriched categories which is highly relevant to the companion paper [7] in which we use pullbacks and pseudo pullbacks to compose strategies with parallel causes. Let $f : A \rightarrow C$ and $g : B \rightarrow C$ be two maps in an Equiv-enriched category. A *pseudo pullback* of f and g is an object D and maps $p : D \rightarrow A$ and $q : D \rightarrow B$ such that $f \circ p \equiv g \circ q$ which satisfy the further property that for any D' and maps $p' : D' \rightarrow A$ and $q' : D' \rightarrow B$ such that $f \circ p' \equiv g \circ q'$, there is a unique map $h : D' \rightarrow D$ such that $p' = p \circ h$ and $q' = q \circ h$; note the insistence on the last two equalities, rather than just equivalences. There is an obvious weakening of pseudo pullbacks to the situation in which the uniqueness is replaced by uniqueness up to \equiv and the equalities by \equiv – these are simple special cases of bilimits called *bipullbacks*.

Right adjoints in a 2-adjunction preserve pseudo pullbacks whereas right adjoints in a pseudo adjunction are only assured to preserve bipullbacks.

A Coalgebraic Perspective on Probabilistic Logic Programming

Tao Gu

University College London, London, UK
tao.gu.18@ucl.ac.uk

Fabio Zanasi

University College London, London, UK
<http://www.zanasi.com/fabio/>
f.zanasi@ucl.ac.uk

Abstract

Probabilistic logic programming is increasingly important in artificial intelligence and related fields as a formalism to reason about uncertainty. It generalises logic programming with the possibility of annotating clauses with probabilities. This paper proposes a coalgebraic perspective on probabilistic logic programming. Programs are modelled as coalgebras for a certain functor F , and two semantics are given in terms of cofree coalgebras. First, the cofree F -coalgebra yields a semantics in terms of derivation trees. Second, by embedding F into another type G , as cofree G -coalgebra we obtain a “possible worlds” interpretation of programs, from which one may recover the usual distribution semantics of probabilistic logic programming.

2012 ACM Subject Classification Theory of computation; Theory of computation \rightarrow Logic

Keywords and phrases probabilistic logic programming, coalgebraic semantics, distribution semantics

Digital Object Identifier 10.4230/LIPIcs.CALCO.2019.10

Funding *Fabio Zanasi*: partial support from EPSRC grant n. EP/R020604/1.

Acknowledgements Fabio Zanasi acknowledges support from EPSRC grant n. EP/R020604/1. The authors thank Alessandro Facchini for useful pointers to the literature, and the anonymous reviewers for the useful comments and feedback.

1 Introduction

Probabilistic logic programming (PLP) [23, 5, 25] is a family of approaches extending the declarative paradigm of logic programming with the possibility of reasoning about uncertainty. This has been proven useful in various applications, including bioinformatics [6, 22], robotics [27] and the semantic web [29].

The most common version of PLP – on which for instance **ProbLog** is based [6], the probabilistic analogue of **Prolog** – is defined by letting clauses in programs to be annotated with mutually independent probabilities. As for the interpretation, *distribution semantics* [25] is typically used as a benchmark for the various implementations of PLP, such as **pD**, **PRISM** and **ProbLog** [24]. In this semantics, the probability of refuting a goal in a program is obtained as a sum of the probabilities of the *possible worlds* (sets of clauses) in which the goal is refutable. The distribution semantics is particularly interesting because it is compatible with the encoding of Bayesian networks as probabilistic logic programs [24], thus indicating that PLP can be effectively employed for Bayesian reasoning.

The main goal of this work is to present a coalgebraic perspective on PLP and its distribution semantics. We first consider the case of ground programs, that is, those without variables. Our approach is based on the observation – inspired by the coalgebraic treatment of “pure” logic programming [16] – that ground programs are in 1-1 correspondence with coalgebras



© Tao Gu and Fabio Zanasi;

licensed under Creative Commons License CC-BY

8th Conference on Algebra and Coalgebra in Computer Science (CALCO 2019).

Editors: Markus Roggenbach and Ana Sokolova; Article No. 10; pp. 10:1–10:21

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

for the functor $\mathcal{M}_{pr}\mathcal{P}_f$, where \mathcal{M}_{pr} is the finite multiset functor on $[0, 1]$ and \mathcal{P}_f is the finite powerset functor. We then provide two coalgebraic semantics for ground PLP.

- The first interpretation $\llbracket - \rrbracket$ is in terms of execution trees called *stochastic derivation trees*, which represent parallel SLD-derivations of a program on a goal. Stochastic derivation trees are the elements of the cofree $\mathcal{M}_{pr}\mathcal{P}_f$ -coalgebra on a given set of atoms At , meaning that any goal $A \in \text{At}$ can be given a semantics in terms of the corresponding stochastic derivation tree by the universal property map $\llbracket - \rrbracket$ to the cofree coalgebra.
- The second interpretation $\langle\langle - \rangle\rangle$ recovers the usual distribution semantics of PLP. This requires some work, as expressing probability distributions on the possible worlds needs a different coalgebra type. We introduce *distribution trees*, a tree-like representation of the distribution semantics, as the elements of the cofree $\mathcal{D}_{\leq 1}\mathcal{P}_f\mathcal{P}_f$ -coalgebra on At , where $\mathcal{D}_{\leq 1}$ is the sub-probability distribution monad. In order to characterise $\langle\langle - \rangle\rangle$ as the map given by universal property of distribution trees, we need a canonical extension of PLP to the setting of $\mathcal{D}_{\leq 1}\mathcal{P}_f\mathcal{P}_f$ -coalgebras. This is achieved via a “possible worlds” natural transformation $\mathcal{M}_{pr}\mathcal{P}_f \Rightarrow \mathcal{D}_{\leq 1}\mathcal{P}_f\mathcal{P}_f$.

In the second part of the paper we recover the same framework for arbitrary probabilistic logic programs, possibly including variables. The encoding of programs as coalgebras is subtler. The space of atoms is now a presheaf indexed by a “Lawvere theory” of terms and substitutions. The coalgebra map can be defined in different ways, depending on the substitution mechanism on which one wants to base resolution. For pure logic programs, the definition by term matching is the best studied, with [17] observing that moving from sets to posets is required in order for the corresponding coalgebra map to be well-defined as a natural transformation between presheaves. A different route is taken in [3], where the problem of naturality is neutralised via “saturation”, a categorical construction which amounts to defining resolution by unification instead of term-matching.

In developing a coalgebraic treatment of PLP with variables, we follow the saturation route, as it also allows to recover the term-matching approach, via “desaturation” [3]. This provides a cofree coalgebra semantics $\llbracket - \rrbracket$ for arbitrary PLP programs, as a rather straightforward generalisation of the saturated semantics of pure logic programs. On the other hand, extending the ground distribution semantics $\langle\langle - \rangle\rangle$ to arbitrary PLP programs poses some challenges: we need to ensure that, in computing the distribution over possible worlds associated to each sub-goal in the computation, each clause of the program is “counted” only once. This is solved by tweaking the coalgebra type of the distribution trees for arbitrary PLP programs, so that some nodes are labelled with clauses of the program. Thanks to this additional information, the term-matching distribution semantics of an arbitrary PLP goal is computable from its distribution tree.

In light of the coalgebraic treatment of pure logic programming [16, 17, 2, 3], the generalisation to PLP may not appear so surprising. In fact, we believe its importance is two-fold. First, whereas the derivation semantics $\llbracket - \rrbracket$ is a straight generalisation of the pure setting, the distribution semantics $\langle\langle - \rangle\rangle$ is genuinely novel, and does not have counterparts in pure logic programming. Second, a paper dedicated to establishing the foundations of coalgebraic PLP is a necessary preliminary step for a number of interesting applications:

- as mentioned, reasoning in Bayesian networks can be seen as a particular case of PLP, equipped with the distribution semantics. Our coalgebraic perspective thus readily applies to Bayesian reasoning, paving the way for combination with recent works [11, 12, 4] modelling belief revision, causal inference and other Bayesian tasks in algebraic terms.
- the combination of logic programming and probabilities comes in different flavours [24]: the more abstract viewpoint offered by coalgebra may provide a unifying perspective on

these approaches, as well as a formal connection with seemingly related languages such as weighted logic programming [8] and Bayesian logic programming [13].

- the coalgebraic treatment of pure logic programming has been used as a formal justification [19, 14] for coinductive logic programming [15, 9]. Coinduction in the context of probabilistic logic programs is, to the best of our knowledge, a completely unexplored field, for which the current paper establishes semantic foundations.

We leave the exploration of these venues as follow-up work.

2 Preliminaries

Signature, Terms, and Categories. A *signature* Σ is a set of function symbols, each equipped with a fixed finite arity. Throughout this paper we fix a signature Σ , and a countably infinite set of variables $Var = \{x_1, x_2, \dots\}$. The Σ -terms over Var are defined as usual. A *context* is a finite set of variables $\{x_1, x_2, \dots, x_n\}$. With some abuse of notation, we shall often use n to denote this context. We say a Σ -term t is *compatible* with context n if the variables appearing in t are all contained in $\{x_1, \dots, x_n\}$.

We are going to reason about Σ -terms categorically using Lawvere theories. First, we will use $\mathbf{Ob}(\mathbf{C})$ to denote the set of objects and $\mathbf{C}[C, D]$ for the set of morphisms $C \rightarrow D$ in a category \mathbf{C} . A \mathbf{C} -indexed *presheaf* is a functor $F: \mathbf{C} \rightarrow \mathbf{Sets}$. \mathbf{C} -indexed presheaves and natural transformations between them form a category $\mathbf{Sets}^{\mathbf{C}}$. Recall that the (opposite) Lawvere Theory of Σ is the category $\mathbf{L}_{\Sigma}^{\text{op}}$ with objects the natural numbers and morphisms $\mathbf{L}_{\Sigma}^{\text{op}}[n, m]$ the n -tuples $\langle t_1, \dots, t_n \rangle$, where each t_i is a Σ -term in context m . For modelling logic programming, it is convenient to think of each $n \in \mathbf{Ob}(\mathbf{L}_{\Sigma}^{\text{op}})$ as representing the context $\langle x_1, \dots, x_n \rangle$, and a morphism $\langle t_1, \dots, t_n \rangle: n \rightarrow m$ as the substitution transforming Σ -terms in context n to Σ -terms in context m by replacing each x_i with t_i . We shall also refer to $\mathbf{L}_{\Sigma}^{\text{op}}$ morphisms simply as substitutions (notation $\theta, \tau, \sigma, \dots$).

Logic programming. We now recall the basics of (pure) logic programming, and refer the reader to [20] for a more systematic exposition. An *alphabet* \mathcal{A} consists of a signature Σ , a set of variables Var , and a set of predicate symbols $\{P_1, P_2, \dots\}$, each with a fixed arity. Given an n -ary predicate symbol P in \mathcal{A} , and Σ -terms t_1, \dots, t_n , $P(t_1 \cdots t_n)$ is called an *atom* over \mathcal{A} . We use A, B, \dots to denote atoms. Given an atom A in context n , and a substitution $\theta = \langle t_1, \dots, t_n \rangle: n \rightarrow m$, we write $A\theta$ for *substitution instance* of A obtained by replacing each appearance of x_i with t_i in A . For convenience, we also use $\{B_1, \dots, B_k\}\theta$ as a shorthand for $\{B_1\theta, \dots, B_k\theta\}$. Given two atoms A and B (over \mathcal{A}), a *unifier* of A and B is a pair $\langle \sigma, \tau \rangle$ of substitutions such that $A\sigma = B\tau$. *Term matching* is a special case of unification, where σ is the identity substitution. In this case we say that τ matches B with A if $A = B\tau$.

A (pure) logic program \mathbb{L} consists of a finite set of clauses \mathcal{C} in the form $H \leftarrow B_1, \dots, B_k$, where H, B_1, \dots, B_k are atoms. H is called the *head* of \mathcal{C} , and B_1, \dots, B_k form the *body* of \mathcal{C} . We denote H by $\text{Head}(\mathcal{C})$, and $\{B_1, \dots, B_k\}$ by $\text{Body}(\mathcal{C})$. A *goal* is simply an atom. Since one can regard a clause $H \leftarrow B_1, \dots, B_k$ as the logic formula $B_1 \wedge \cdots \wedge B_k \rightarrow H$, we say that a goal G is *derivable* in \mathbb{L} if there exists a derivation of G with empty assumption using the clauses in \mathbb{L} .

The central task of logic programming is to check whether a goal G is *provable* (or *refutable* as in some literature) in a program \mathbb{L} , in the sense that some substitution instance of G is derivable in \mathbb{L} . The key algorithm for this task is SLD-resolution, see e.g. [20]. We use the notation $\mathbb{L} \vdash G$ to mean that G is provable in \mathbb{L} .

Probabilistic logic programming. We now recall the basics of PLP; the reader may consult [7, 6] for a more comprehensive introduction. A probabilistic logic program \mathbb{P} based on a logic program \mathbb{L} assigns a probability label r to each clause \mathcal{C} in \mathbb{L} , denoted as $\text{Label}(\mathcal{C})$. One may also regard \mathbb{P} as a set of probabilistic clauses of the form $r :: \mathcal{C}$, where \mathcal{C} is a clause in \mathbb{L} , and each clause \mathcal{C} is assigned a unique probability label r in \mathbb{P} . We also refer to $r :: \mathcal{C}$ simply as clauses.

► **Example 1.** As our leading example we introduce the following probabilistic logic program \mathbb{P}^{al} . It models the scenario of Mary’s house alarm, which is supposed to detect burglars, but it may be accidentally triggered by an earthquake. Mary may hear the alarm if she is awake, but even if the alarm is not sounding, in case she experiences an auditory hallucination (paracusia). The language of \mathbb{P}^{al} includes 0-ary predicates `Alarm`, `Eearthquake`, `Burglary`, and 1-ary predicates `Wake(-)`, `Hear_alarm(-)` and `Paracusia(-)`, and signature $\Sigma_{al} = \{\text{Mary}^0\}$ consisting of a constant. We do not have variables here, so \mathbb{P}^{al} is a ground program. For readability we abbreviate `Mary` as `M` in the program.

| | |
|---|--|
| 0.01 :: Earthquake ← 0.2 :: Burglary ← 0.5 :: Alarm ← Earthquake 0.9 :: Alarm ← Burglary | 0.01 :: Paracusia(M) ← 0.6 :: Wake(M) ← 0.8 :: Hear_alarm(M) ← Alarm, Wake(M) 0.3 :: Hear_alarm(M) ← Paracusia(M) |
|---|--|

As a generalisation of the pure case, in probabilistic logic programming one is interested in the *probability* of a goal G being refutable in a program \mathbb{P} . There are potentially multiple ways to define such probability – in this paper we focus on the *distribution semantics* [7].

The probability of refuting a goal is computed in the distribution semantics as follows. Given a probabilistic logic program $\mathbb{P} = \{p_1 :: \mathcal{C}_1, \dots, p_n :: \mathcal{C}_n\}$, let $|\mathbb{P}|$ be its underlying pure logic program, namely $|\mathbb{P}| = \{\mathcal{C}_1, \dots, \mathcal{C}_n\}$. A *sub-program* \mathbb{L} of $|\mathbb{P}|$ is a logic program consisting of a subset of the clauses in $|\mathbb{P}|$. This justifies using $\mathcal{P}(|\mathbb{P}|)$ to denote the set of all sub-programs of $|\mathbb{P}|$, and using $\mathbb{L} \subseteq |\mathbb{P}|$ to denote that \mathbb{L} is a sub-program of \mathbb{P} . The central concept of the distribution semantics is that \mathbb{P} determines a distribution $\mu_{\mathbb{P}}$ over the sub-programs $\mathcal{P}(|\mathbb{P}|)$: for any $\mathbb{L} \in \mathcal{P}(|\mathbb{P}|)$, $\mu_{\mathbb{P}}(\mathbb{L}) := \prod_{\mathcal{C}_i \in \mathbb{L}} p_i \prod_{\mathcal{C}_j \in |\mathbb{P}| \setminus \mathbb{L}} (1 - p_j)$. The value $\mu_{\mathbb{P}}(\mathbb{L})$ is called the *probability* of the sub-program \mathbb{L} . For an arbitrary goal $G \in \text{At}$, the *success probability* $\text{Pr}_{\mathbb{P}}(G)$ of G w.r.t. program \mathbb{P} is then defined as the sum of the probabilities of all the sub-programs of \mathbb{P} in which G is refutable:

$$\text{Pr}_{\mathbb{P}}(G) := \sum_{|\mathbb{P}| \supseteq \mathbb{L} \vdash G} \mu_{\mathbb{P}}(\mathbb{L}) = \sum_{|\mathbb{P}| \supseteq \mathbb{L} \vdash G} \left(\prod_{\mathcal{C}_i \in \mathbb{L}} p_i \prod_{\mathcal{C}_j \in |\mathbb{P}| \setminus \mathbb{L}} (1 - p_j) \right) \quad (1)$$

Intuitively one can regard every clause in \mathbb{P} as an event, then every sub-program \mathbb{L} can be seen as a possible world, and $\mu_{\mathbb{P}}$ is a distribution over the possible worlds.

► **Example 2.** For the program \mathbb{P}^{al} , consider the goal `Hear_alarm(M)`. By definition (1), we can compute its success probability $\text{Pr}_{\mathbb{P}^{al}}(\text{Hear_alarm}(\text{M}))$, and the result is 0.091102896.

3 Ground case

In this section we introduce a coalgebraic semantics for *ground* probabilistic logic programming, i.e. for those programs where no variable appears. Our approach consists of two parts. First, in Subsection 3.1, we represent PLP logic programs as coalgebras and their executions as a final coalgebra semantics (Subsection 3.2) – this is a straight generalisation of the coalgebraic treatment of pure logic programs given in [16]. Next, in Subsection 3.3 we show

how to represent the distribution semantics in terms as a final coalgebra, via a transformation of the coalgebra type of logic programs. Appendix A shows how the probability of a goal is effectively computable from the above representation.

3.1 Coalgebraic Representation of PLP

A ground program will be represented as a coalgebra for the composite $\mathcal{M}_{pr}\mathcal{P}_f: \mathbf{Sets} \rightarrow \mathbf{Sets}$ of the finite probability functor $\mathcal{M}_{pr}: \mathbf{Sets} \rightarrow \mathbf{Sets}$ and the finite powerset functor $\mathcal{P}_f: \mathbf{Sets} \rightarrow \mathbf{Sets}$. The definition of \mathcal{M}_{pr} deserves some further explanation. It can be seen as the finite multiset functor based on the commutative monoid $([0, 1], 0, \vee)$, where $a \vee b := 1 - (1 - a)(1 - b)$. That is to say, on objects, $\mathcal{M}_{pr}(A)$ is the set of all *finite probability assignments* $\varphi: A \rightarrow [0, 1]$ with a finite support $\text{supp}(\varphi) := \{a \in A \mid \varphi(a) \neq 0\}$. For φ with support $\{a_1, \dots, a_k\}$ and values $\varphi(a_i) = r_i$, it will often be convenient to use the standard notation $\varphi = \sum_{i=1}^k r_i a_i$ or $\varphi = r_1 a_1 + \dots + r_k a_k$, where the purely formal “+” here should not be confused with the addition in \mathbb{R} . On morphisms, $\mathcal{M}_{pr}(h: A \rightarrow B)$ maps $\sum_{i=1}^k r_i a_i$ to $\sum_{i=1}^k r_i h(a_i)$.

Fix a ground probabilistic logic program \mathbb{P} on a set of ground atoms At . The definition of \mathbb{P} can be encoded as an $\mathcal{M}_{pr}\mathcal{P}_f$ -coalgebra $p: \text{At} \rightarrow \mathcal{M}_{pr}(\mathcal{P}_f(\text{At}))$, as follows. Given $A \in \text{At}$,

$$p(A): \quad \mathcal{P}_f(\text{At}) \quad \rightarrow \quad [0, 1]$$

$$\{B_1, \dots, B_n\} \quad \mapsto \quad \begin{cases} r & \text{if } r :: A \leftarrow B_1, \dots, B_n \text{ is a clause in } \mathbb{P} \\ 0 & \text{otherwise.} \end{cases}$$

Or, equivalently, $p(A) := \sum_{(r::A \leftarrow B_1, \dots, B_n) \in \mathbb{P}} r \{B_1, \dots, B_n\}$.

► **Example 3.** Consider program \mathbb{P}^{al} from Example 1. The set of ground atoms At_{al} is $\{\text{Alarm}, \text{Earthquake}, \text{Burgary}, \text{Wake}(\text{M}), \text{Paracusia}(\text{M}), \text{Hear_alarm}(\text{M})\}$. Here are some values of the corresponding coalgebra $p_{al}: \text{At}_{al} \rightarrow \mathcal{M}_{pr}\mathcal{P}_f\text{At}_{al}$:

$$p_{al}(\text{Hear_alarm}(\text{M})) = 0.8\{\text{Alarm}, \text{Wake}(\text{M})\} + 0.3\{\text{Paracusia}(\text{M})\} \quad p_{al}(\text{Earthquake}) = 0.01\{\}$$

► **Remark 4.** One might wonder why not simply adopt $\mathcal{P}_f(\mathcal{P}_f(-) \times [0, 1])$ as the coalgebra type for PLP. Note that this encoding would not have 1 – 1 correspondence with ground PLP programs: a clause $\mathcal{C} \in \mathcal{P}_f(\text{At})$ may be associated with different probabilities in $[0, 1]$, which violates the standard definition of PLP programs.

3.2 Derivation Semantics

In this section we are going to construct the final $\text{At} \times \mathcal{M}_{pr}\mathcal{P}_f(-)$ -coalgebra, thus providing a semantic interpretation for probabilistic logic programs based on At .

Before the technical developments, we give an intuitive view on the semantics that the final coalgebra is going to provide. We shall represent each goal as a *stochastic derivation tree* in the final coalgebra. These trees are the probabilistic version of and-or derivation trees, which represent parallel SLD-resolutions in pure logic programming [10].

► **Definition 5 (Stochastic derivation trees).** *Given a ground PLP program \mathbb{P} and a ground atom A , the stochastic derivation tree for A in \mathbb{P} is the possibly infinite tree \mathcal{T} such that:*

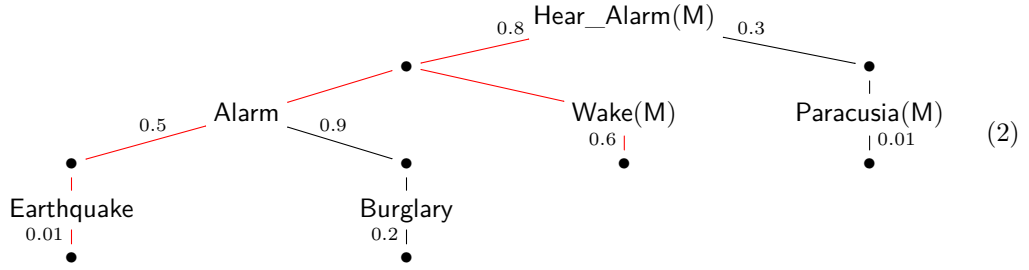
1. *Every node is either an atom-node (labelled with an atom $A' \in \text{At}$) or a clause-node (labelled with \bullet). They appear alternately in depth, in this order. The root is an atom-node labelled with A .*

10:6 A Coalgebraic Perspective on Probabilistic Logic Programming

2. Each edge from an atom-node to its (clause-)children is labelled with a probability value.
3. Suppose s is an atom-node with label A' . Then for every clause $r :: A' \leftarrow B_1, \dots, B_k$ in \mathbb{P} , s has exactly one child t , the edge $s \rightarrow t$ is labelled with r , and t has exactly k children labelled with B_1, \dots, B_k , respectively.

The final coalgebra semantics $\llbracket - \rrbracket_p$ for a program \mathbb{P} will map a goal A to the stochastic derivation tree representing all possible SLD-resolutions of A in \mathbb{P} .

► **Example 6.** Continuing Example 1, $\llbracket \text{Hear_alarm}(\text{M}) \rrbracket_{p_{al}}$ is the stochastic derivation tree below. The subtree highlighted in red represents one of the successful refutations of $\text{Hear_alarm}(\text{M})$ in p_{al} : indeed, note that a single child is selected for each atom-node A (corresponding to a clause matching A), all children of any clause-node are selected (corresponding to the atoms in the body of the clause), and the subtree has clause-nodes as leaves (all atoms are proven).



Any such subtree describes a refutation, but does not yield a probability value to be associated to a goal – this is the remit of the distribution semantics, see Example 10 below.

In the remaining part of the section, we construct the cofree coalgebra for $\mathcal{M}_{pr}\mathcal{P}_f$ via a so-called terminal sequence [28], and obtain $\llbracket - \rrbracket_p$ from the resulting universal property. We report the steps of the terminal sequence as they are instrumental in showing that the elements of the cofree coalgebra can be seen as stochastic derivation trees.

► **Construction 7.** The terminal sequence for the functor $\text{At} \times \mathcal{M}_{pr}\mathcal{P}_f(-) : \mathbf{Sets} \rightarrow \mathbf{Sets}$ consists of sequences of objects $\{X_\alpha\}_{\alpha \in \mathbf{Ord}}$ and arrows $\{\delta_\beta^\alpha : X_\alpha \rightarrow X_\beta\}_{\beta < \alpha \in \mathbf{Ord}}$ constructed by the following inductive definitions:

$$X_\alpha := \begin{cases} \text{At} & \alpha = 0 \\ \text{At} \times \mathcal{M}_{pr}\mathcal{P}_f(X_\xi) & \alpha = \xi + 1 \\ \lim\{\delta_\xi^\chi \mid \xi < \chi < \alpha\} & \alpha \text{ is limit} \end{cases} \quad \delta_\beta^\alpha := \begin{cases} \pi_1 & \alpha = 1, \beta = 0 \\ \text{id}_{\text{At}} \times \mathcal{M}_{pr}\mathcal{P}_f(\delta_\xi^{\xi+1}) & \alpha = \beta + 1 = \xi + 2 \\ \text{the limit projections} & \alpha \text{ is limit, } \beta < \alpha \\ \text{universal map to } X_\beta & \beta \text{ is limit, } \alpha = \beta + 1 \end{cases}$$

► **Proposition 8.** The terminal sequence for the functor $\text{At} \times \mathcal{M}_{pr}\mathcal{P}_f(-)$ converges to a limit X_γ such that $X_\gamma \cong X_{\gamma+1}$.

Proof. We need to verify the assumptions of [28, Corollary 3.3]. It is well-known that \mathcal{P}_f is ω -accessible, and \mathcal{M}_{pr} has the same property, see e.g. [26, Prop. 6.1.2]. Because accessibility is defined in terms of colimit preservation, it is clearly preserved by composition, and thus $\mathcal{M}_{pr}\mathcal{P}_f$ is also accessible. It remains to check that it preserves monics. For \mathcal{M}_{pr} , given any monomorphism $i : C \rightarrow D$ in \mathbf{Sets} , suppose $\mathcal{M}_{pr}(i)(\varphi) = \mathcal{M}_{pr}(i)(\varphi')$ for some $\varphi, \varphi' \in \mathcal{M}_{pr}(C)$. Then for any $d \in D$, $\mathcal{M}_{pr}(i)(\varphi)(d) = \mathcal{M}_{pr}(i)(\varphi')(d)$. If we focus on the image $i[C]$, then there is an inverse function $i^{-1} : i[C] \rightarrow C$, and $\mathcal{M}_{pr}(i)(\varphi) = \mathcal{M}_{pr}(i)(\varphi')$ implies that $\varphi(i^{-1}(d)) = \varphi'(i^{-1}(d))$ for any $d \in i[C]$. But this simply means that $\varphi = \varphi'$. As

the same is true for \mathcal{P}_f and the property is preserved by composition, we have that $\mathcal{M}_{pr}\mathcal{P}_f$ preserves monics. We can then conclude by [28, Corollary 3.3] that the terminal sequence for $\text{At} \times \mathcal{M}_{pr}\mathcal{P}_f$ converges to the cofree $\mathcal{M}_{pr}\mathcal{P}_f$ -coalgebra on At . ◀

Note that $X_{\gamma+1}$ is defined as $\text{At} \times \mathcal{M}_{pr}\mathcal{P}_f(X_\gamma)$, and the above isomorphism makes $X_\gamma \rightarrow \text{At} \times \mathcal{M}_{pr}\mathcal{P}_f(X_\gamma)$ the final $\text{At} \times \mathcal{M}_{pr}\mathcal{P}_f$ -coalgebra – or, in other words, *cofree $\mathcal{M}_{pr}\mathcal{P}_f$ -coalgebra* on At . As for the tree representation of the elements of X_γ , recall that elements of the cofree $\mathcal{P}_f\mathcal{P}_f$ -coalgebra on At can be seen as and-or trees [16]. By replacing the first \mathcal{P}_f with \mathcal{M}_{pr} , effectively one adds probability values to the edges from and-nodes to or-nodes (which are edges from atom-nodes to or-nodes in our stochastic derivation trees), as in (2). Thus stochastic derivation trees as in Definition 5 are elements of X_γ . The action of the coalgebra map $\cong: X_\gamma \rightarrow \text{At} \times \mathcal{M}_{pr}\mathcal{P}_f(X_\gamma)$ is best seen with an example: the tree \mathcal{T} in (2) (an element of X_γ) is mapped to the pair $\langle \text{Hear_alarm}(\text{M}), \varphi \rangle$, where φ is the function $\mathcal{P}_f(X_\gamma) \rightarrow [0, 1]$ assigning 0.8 to the set consisting of the subtrees of \mathcal{T} with root Alarm and with root $\text{Wake}(\text{M})$, 0.3 to the singleton consisting to the subtree of \mathcal{T} with root $\text{Paracusia}(\text{M})$, and 0 to any other finite set of trees.

With all the definitions at hand, it is straightforward to check that $\llbracket - \rrbracket_p$ mapping $A \in \text{At}$ to its stochastic derivation tree in p makes the following diagram commute

$$\begin{array}{ccc} \text{At} & \xrightarrow{\llbracket - \rrbracket_p} & X_\gamma \\ \downarrow \langle id, p \rangle & & \downarrow \cong \\ \text{At} \times \mathcal{M}_{pr}\mathcal{P}_f(\text{At}) & \xrightarrow{id \times \mathcal{M}_{pr}\mathcal{P}_f(\llbracket - \rrbracket_p)} & \text{At} \times \mathcal{M}_{pr}\mathcal{P}_f(X_\gamma) \end{array}$$

and thus by uniqueness it coincides with the $\text{At} \times \mathcal{M}_{pr}\mathcal{P}_f$ -coalgebra map provided by the universal property of the final $\text{At} \times \mathcal{M}_{pr}\mathcal{P}_f$ -coalgebra $X_\gamma \rightarrow \text{At} \times \mathcal{M}_{pr}\mathcal{P}_f(X_\gamma)$.

3.3 Distribution Semantics

This section gives a coalgebraic definition of the usual *distribution semantics* of probabilistic logic programming. As in the previous section, before the technical developments we gather some preliminary intuition. Recall from Section 2 that the core of the distribution semantics is the probability distribution over the sub-programs (sets of clauses) of a given program \mathbb{P} . These sub-programs are also called (possible) worlds, and the distribution semantics of a goal is the sum of the probabilities of all the worlds in which it is refutable.

In order to code this information as elements of a final coalgebra, we need to present it in tree-shape. Roughly speaking, we form a distribution over the sub-programs along the execution tree. This justifies the following notion of *distribution trees*.

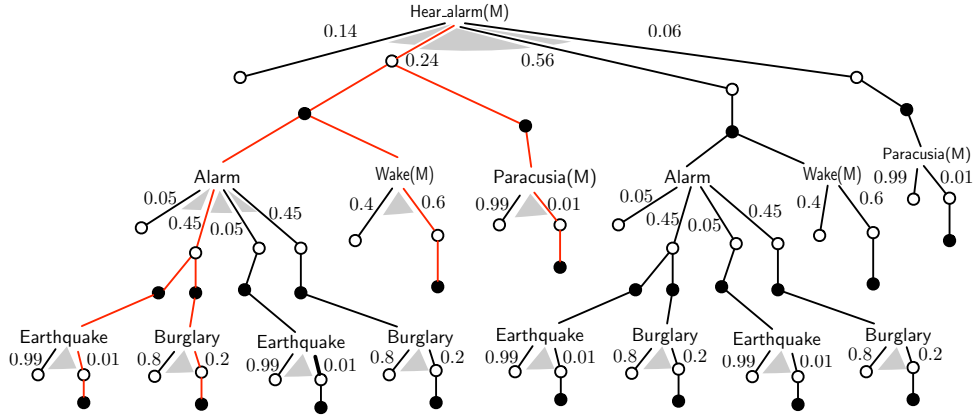
► **Definition 9** (Distribution trees). *Given a PLP program \mathbb{P} and an atom A , the distribution tree for A in \mathbb{P} is the possibly infinite tree \mathcal{T} satisfying the following properties:*

1. *Every node is exactly one of the three kinds: atom-node (labelled with an atom $A \in \text{At}$), world-node (labelled with \circ), clause-node (labelled with \bullet). They appear alternatingly in this order in depth. The root is an atom-node labelled with A .*
2. *Every edge from an atom-node to its (world-)children is labelled with a probability value, and they sum up to one.*
3. *Suppose s is an atom-node labelled with A' , and $C = \{C_1, \dots, C_m\}$ is the set of all the clauses in \mathbb{P} whose head is A' . Then s has 2^m children, each standing for a subset X of C . If a child t stands for X , then the edge $s \rightarrow t$ is labelled with probability $\prod_{C \in X} \text{Label}(C) \cdot \prod_{C' \in C \setminus X} (1 - \text{Label}(C'))$ – recall that $\text{Label}(C)$ is the probability labelling*

\mathcal{C} . Also, t has exactly $|X|$ children, each standing for a clause $C \in X$. If a child u stands for $C = r :: A' \leftarrow B_1, \dots, B_k$, then u has k children, labelled with B_1, \dots, B_k respectively.

Comparing distribution trees with stochastic derivation trees (Definition 5), one can observe the addition of another class of nodes, representing possible worlds. Moreover, the possible worlds associated with an atom-node (a goal) must form a probability distribution – as opposed to stochastic derivation trees, in which probabilities labelling parallel edges do not need to share any relationship. An example of the distribution tree associated with a goal is provided in the continuation of our leading example (Examples 1 and 6).

► **Example 10.** In the context of Example 1, the distribution tree of $\text{Hear_alarm}(M)$ is depicted below, where we use grey shades to emphasise sets of edges expressing a probability distribution. Also, note the \oslash s with no children, standing for empty worlds.



In the literature, the distribution semantics usually associates with a goal a single probability value (1), rather than a whole tree. However, given the distribution tree it is straightforward to compute such probability. The subtree highlighted in red above describes a refutation of $\text{Hear_alarm}(M)$ with probability 0.000001296 (= the product of all the probabilities in the subtree). The sum of all the probabilities associated to such “refutation” subtrees yields the usual distribution semantics (1) – the computation is shown in detail in Appendix A.

In the remainder of this section, we focus on the coalgebraic characterisation of distribution trees and the associated semantics map. Our strategy will be to introduce a novel coalgebra type $\mathcal{D}_{\leq 1}\mathcal{P}_f\mathcal{P}_f$, such that distribution trees can be seen as elements of the cofree coalgebra. Then, we will provide a natural transformation $\mathcal{M}_{pr} \Rightarrow \mathcal{D}_{\leq 1}\mathcal{P}_f$, which can be used to transform stochastic derivation trees into distribution trees. Finally, composing the universal properties of these cofree coalgebras will yield the desired distribution semantics.

We begin with the definition of $\mathcal{D}_{\leq 1}\mathcal{P}_f$. This is simply the composite $\mathcal{D}_{\leq 1}\mathcal{P}_f : \mathbf{Sets} \rightarrow \mathbf{Sets}$, where $\mathcal{D}_{\leq 1}$ is the *sub-probability distribution* functor. Recall that $\mathcal{D}_{\leq 1}$ maps X to the set of sub-probability distributions with finite supports on X (i.e., convex combinations of elements of X whose sum is less or equal to 1), and acts component-wise on functions.

► **Remark 11.** Note that we cannot work with full probabilities here, since a goal may not match any clause. In such a case there is no world in which the goal is refutable and its probability in the program is 0.

The next step is to recover distribution trees as the elements of the $\mathcal{D}_{\leq 1}\mathcal{P}_f\mathcal{P}_f$ -cofree coalgebra on At . This goes via a terminal sequence, similarly to the case of $\mathcal{M}_{pr}\mathcal{P}_f$ in the previous section. The terminal sequence for $\text{At} \times \mathcal{D}_{\leq 1}\mathcal{P}_f\mathcal{P}_f(-) : \mathbf{Sets} \rightarrow \mathbf{Sets}$ is constructed as the one for $\text{At} \times \mathcal{M}_{pr}\mathcal{P}_f(-) : \mathbf{Sets} \rightarrow \mathbf{Sets}$ (Construction 7), with $\mathcal{D}_{\leq 1}\mathcal{P}_f$ replacing \mathcal{M}_{pr} .

► **Proposition 12.** *The terminal sequence of $\text{At} \times \mathcal{D}_{\leq 1} \mathcal{P}_f \mathcal{P}_f(-)$ converges at some limit ordinal χ , and $(\lambda_\chi^{x+1})^{-1} : Y_\chi \rightarrow \text{At} \times \mathcal{D}_{\leq 1} \mathcal{P}_f \mathcal{P}_f Y_\chi$ is the final $\text{At} \times \mathcal{D}_{\leq 1} \mathcal{P}_f \mathcal{P}_f$ coalgebra.*

Proof. As for Proposition 8, by [28, Cor. 3.3] it suffices to show that $\mathcal{D}_{\leq 1} \mathcal{P}_f$ is accessible and preserves monos. Both are simple exercises; in particular, see [1] for accessibility of $\mathcal{D}_{\leq 1}$. ◀

The association of distribution trees with elements of Y_χ is suggested by the type $\text{At} \times \mathcal{D}_{\leq 1} \mathcal{P}_f \mathcal{P}_f$. Indeed, $\text{At} \times \mathcal{D}_{\leq 1}$ is the layer of atom-nodes, labelled with elements of At and with outgoing edges forming a sub-probability distribution; the first \mathcal{P}_f is the layer of world-nodes; the second \mathcal{P}_f is the layer of clause-nodes. The coalgebra map $Y_\chi \rightarrow \text{At} \times \mathcal{D}_{\leq 1} \mathcal{P}_f \mathcal{P}_f Y_\chi$ associates a goal to subtrees of its distribution trees, analogously to the coalgebra structure on stochastic derivation trees in the previous section.

The last ingredient we need is a translation of stochastic derivation trees into distribution trees. We formalise this as a natural transformation $\text{pw} : \mathcal{M}_{pr} \Rightarrow \mathcal{D}_{\leq 1} \mathcal{P}_f$. The naturality of pw can be checked with a simple calculation.

► **Definition 13.** *The “possible worlds” natural transformation $\text{pw} : \mathcal{M}_{pr} \Rightarrow \mathcal{D}_{\leq 1} \mathcal{P}_f$ is defined by $\text{pw}_X : \varphi \mapsto \sum_{Y \subseteq \text{supp}(\varphi)} r_Y Y$, where each $r_Y = \prod_{y \in Y} \varphi(y) \cdot \prod_{y' \in \text{supp}(\varphi) \setminus Y} (1 - \varphi(y'))$. In particular, when $\text{supp}(\varphi)$ is empty, $\text{pw}_X(\varphi) = 0$.*

Now we have all the ingredients to characterise the distribution semantics coalgebraically, as the morphism $\llbracket - \rrbracket_p : \text{At} \rightarrow Y_\chi$ defined by the following diagram, which maps $A \in \text{At}$ to its distribution tree in p .

$$\begin{array}{ccccc}
 \text{At} & \xrightarrow{\llbracket - \rrbracket_p} & X_\gamma & \xrightarrow{!} & Y_\chi \\
 \downarrow \langle \text{id}_{\text{At}}, p \rangle & \dashrightarrow & \downarrow \cong & & \downarrow \cong \\
 \text{At} \times \mathcal{M}_{pr} \mathcal{P}_f \text{At} & \xrightarrow{\text{id}_{\text{At}} \times \mathcal{M}_{pr} \mathcal{P}_f (\llbracket - \rrbracket_p)} & \text{At} \times \mathcal{M}_{pr} \mathcal{P}_f X_\gamma & & \\
 \downarrow \text{id}_{\text{At}} \times \text{pw}_{\mathcal{P}_f(\text{At})} & \dashrightarrow & \downarrow \text{id}_{\text{At}} \times \text{pw}_{\mathcal{P}_f X_\gamma} & & \\
 \text{At} \times \mathcal{D}_{\leq 1} \mathcal{P}_f \mathcal{P}_f \text{At} & \xrightarrow{\text{id}_{\text{At}} \times \mathcal{D}_{\leq 1} \mathcal{P}_f \mathcal{P}_f (\llbracket - \rrbracket_p)} & \text{At} \times \mathcal{D}_{\leq 1} \mathcal{P}_f \mathcal{P}_f X_\gamma & \xrightarrow{\text{id}_{\text{At}} \times \mathcal{D}_{\leq 1} \mathcal{P}_f \mathcal{P}_f (!)} & \text{At} \times \mathcal{D}_{\leq 1} \mathcal{P}_f \mathcal{P}_f Y_\chi
 \end{array} \tag{3}$$

Note the use of pw to extend probabilistic logic programs and stochastic derivation trees to the same coalgebra type as distribution trees. Then the distribution semantics $\llbracket - \rrbracket_p$ is uniquely defined by the universal property of the final $\text{At} \times \mathcal{D}_{\leq 1} \mathcal{P}_f \mathcal{P}_f$ -coalgebra. By uniqueness, it can also be computed as the composite $! \circ \llbracket - \rrbracket_p$, that is, first one derives the semantics $\llbracket - \rrbracket_p$, then applies the translation pw to each level of the resulting stochastic derivation tree, in order to turn it into a distribution tree.

4 General Case

We now generalise our coalgebraic treatment to arbitrary probabilistic logic programs and goals, possibly including variables. The section has the same structure as the one devoted to the ground case. First, in Subsection 4.2, we give a coalgebraic representation for general PLP , and equip it with a final coalgebra semantics in terms of stochastic derivation trees. Next, in Subsection 4.3, we study the coalgebraic representation of the distribution semantics. We begin by introducing our leading example – an extension of Example 1.

► **Example 14.** We tweak the ground program of Example 1. Now it is not just Mary that may hear the alarm, but also her neighbours. There is a small probability that the alarm

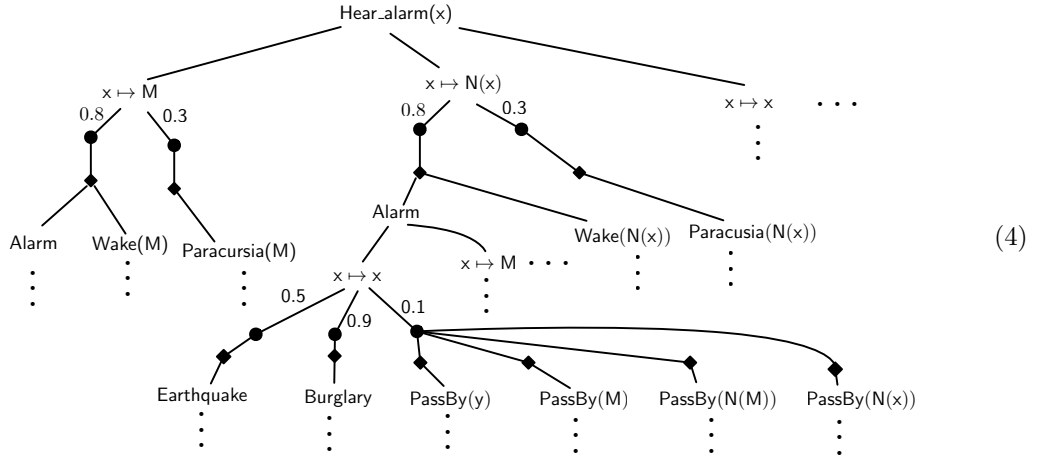
10:10 A Coalgebraic Perspective on Probabilistic Logic Programming

rings because someone passes too close to Mary's house. However, we can only estimate the possibility of paracusia and being awake for Mary, not the neighbours. The revised program, which by abuse of notation we also call \mathbb{P}^{al} , is based on an extension of the language in Example 1: we add a new 1-ary function symbol Neigh^1 to the signature Σ_{al} , and a new 1-ary predicate $\text{PassBy}(-)$ to the alphabet. Note the appearance of a variable x .

| | | | | |
|-------------------------|---|----------------------|---|----------------|
| 0.01 :: Earthquake | ← | 0.5 :: Alarm | ← | Earthquake |
| 0.2 :: Burglary | ← | 0.9 :: Alarm | ← | Burglary |
| 0.6 :: Wake(Mary) | ← | 0.1 :: Alarm | ← | PassBy(x) |
| 0.01 :: Paracusia(Mary) | ← | 0.3 :: Hear_alarm(x) | ← | Paracusia(x) |
| 0.8 :: Wake(Neigh(x)) | ← | 0.8 :: Hear_alarm(x) | ← | Alarm, Wake(x) |

As we want to maintain our approach a direct generalisation of the coalgebraic semantics [3] of pure logic programs, the derivation semantics $\llbracket - \rrbracket$ for PLP will represent resolution by *unification*. This means that, at each step of the computation, given a goal A , one seeks substitutions θ, τ such that $A\theta = H\tau$ for some head H of a clause in the program. As a roadmap, we anticipate the way this computation is represented in terms of stochastic derivation trees (Definition 20 below), with a continuation of our leading example.

► **Example 15.** In the context of Example 14, the tree for $\llbracket \text{Hear_alarm}(x) \rrbracket_{\mathbb{P}^{al}}$ is (partially) depicted below. Compared to the ground case (Example 6), now substitutions applied on the goal side appear explicitly as labels. We abbreviate Neigh as N and Mary as M .



Resolution by unification as above will be implemented in two stages. The first step is devising a map for term-matching. Assuming that the substitution instance $A\theta$ of a goal A is already given, we define p performing term-matching of $A\theta$ in a given program \mathbb{P} :

$$p(A\theta): \{B_1\tau_i, \dots, B_k\tau_i\}_{i \in I \subseteq \mathbb{N}} \mapsto \begin{cases} r & (r :: H \leftarrow B_1, \dots, B_k) \in \mathbb{P} \text{ and} \\ & I \text{ contains all } i \text{ s.t. } A\theta = H\tau_i \\ 0 & \text{otherwise.} \end{cases} \quad (5)$$

Intuitively, one application of such map is represented in a tree structure as Example 15 by the first two layers of the subtree rooted at θ . The reason why the domain of $p(A)$ is a *countable* set $\{B_1\tau_i, \dots, B_k\tau_i\}_{i \in I \subseteq \mathbb{N}}$ of instances of the same body B_1, \dots, B_k is that the same clause may match a goal with countably many different substitutions τ_i . For example in the bottom part of (4) there are countably infinite substitutions τ_i matching the head of $\text{Alarm} \leftarrow \text{PassBy}(x)$ to the goal Alarm , substituting x with Mary , $\text{Neigh}(\text{Mary})$, $\text{Neigh}(x), \dots$

This will be reflected in the coalgebraic representation of PLP (see (7) below) by the use of the countable powset functor \mathcal{P}_c .

In order to model arbitrary unification, the second step is considering all substitutions θ on the goal A such that a term-matcher for $A\theta$ exists. There is an elegant categorical construction [3] packing together these two steps into a single coalgebra map. We will present it in subsection 4.1, and then use it to present the derivation semantics anticipated by Example 15 (Section 4.2). Finally we will give a coalgebraic view on the distribution semantics for PLP (Section 4.3).

4.1 Coalgebraic Representation of PLP

Towards a categorification of general PLP, the first concern is to account for the presence of variables in atoms. This is standardly done by letting the space of atoms on an alphabet \mathcal{A} be a presheaf $\text{At}: \mathbf{L}_\Sigma^{\text{op}} \rightarrow \mathbf{Sets}$ rather than a set. Here the index category $\mathbf{L}_\Sigma^{\text{op}}$ is the opposite *Lawvere Theory* of Σ (see Section 2). For each $n \in \text{Ob}(\mathbf{L}_\Sigma^{\text{op}})$, $\text{At}(n)$ is defined as the set of \mathcal{A} -atoms in context n . Given a n -tuple $\theta = \langle t_1, \dots, t_n \rangle \in \mathbf{L}_\Sigma^{\text{op}}[n, m]$ of Σ -terms in context m , $\text{At}(\theta): \text{At}(n) \rightarrow \text{At}(m)$ is defined by substitution, namely $\text{At}(\theta)(A) = A\theta$, for any $A \in \text{At}(n)$.

As observed in [17] for pure logic programs, if we naively try to model our specification (5) for p as a coalgebra on At , we run into problems: indeed p is not a natural transformation, thus not a morphism between presheaves. Intuitively, this is because the existence of a term-matching for a goal A does not necessarily imply the existence of a term-matching for its substitution instance $A\sigma$. For pure logic programs, this problem can be solved in at least two ways. First, [17] relaxes naturality by changing the base category of presheaves from \mathbf{Sets} to \mathbf{Poset} . We take here the second route, namely give a “saturated” coalgebraic treatment of PLP, generalising the modelling of pure logic programs proposed in [3]. This approach has the advantage of letting us work with \mathbf{Sets} -based presheaves, and be still able to recover term-matching via a “desaturation” operation – see [3] and Appendix B.

The Saturation Adjunction. To this aim, we briefly recall the saturated approach from [3]. The central piece is the adjunction $\mathcal{U} \dashv \mathcal{K}$ on presheaf categories, as on the left below.

$$\begin{array}{ccc}
 \mathbf{Sets}^{\mathbf{L}_\Sigma^{\text{op}}} & \begin{array}{c} \xrightarrow{\mathcal{U}} \\ \perp \\ \xleftarrow{\mathcal{K}} \end{array} & \mathbf{Sets}^{|\mathbf{L}_\Sigma^{\text{op}}|} \\
 & & \begin{array}{ccc} |\mathbf{L}_\Sigma^{\text{op}}| & \xleftarrow{\iota} & \mathbf{L}_\Sigma^{\text{op}} \\ \mathbf{F} \downarrow & \swarrow \mathcal{K}(\mathbf{F}) & \\ \mathbf{Sets} & & \end{array}
 \end{array} \tag{6}$$

Here $|\mathbf{L}_\Sigma^{\text{op}}|$ is the discretisation of $\mathbf{L}_\Sigma^{\text{op}}$, i.e. all the arrows but the identities are dropped. The left adjoint \mathcal{U} is the forgetful functor, given by precomposition with the obvious inclusion $\iota: |\mathbf{L}_\Sigma^{\text{op}}| \rightarrow \mathbf{L}_\Sigma^{\text{op}}$. \mathcal{U} has a right adjoint $\mathcal{K} = \text{Ran}\iota: \mathbf{Sets}^{|\mathbf{L}_\Sigma^{\text{op}}|} \rightarrow \mathbf{Sets}^{\mathbf{L}_\Sigma^{\text{op}}}$, which sends every presheaf $\mathbf{F}: |\mathbf{L}_\Sigma^{\text{op}}| \rightarrow \mathbf{Sets}$ to its *right Kan extension* along ι , as in the rightmost diagram in (6). The definition of \mathcal{K} can be computed [21] as follows:

- on objects $\mathbf{F} \in \text{Ob}(\mathbf{Sets}^{|\mathbf{L}_\Sigma^{\text{op}}|})$, the presheaf $\mathcal{K}(\mathbf{F}): \mathbf{L}_\Sigma^{\text{op}} \rightarrow \mathbf{Sets}$ is defined by letting $\mathcal{K}(\mathbf{F})(n)$ be the product $\mathcal{K}(\mathbf{F})(n) = \prod_{\theta \in \mathbf{L}_\Sigma^{\text{op}}[n, m]} \mathbf{F}(m)$, where m ranges over $\text{Ob}(\mathbf{L}_\Sigma^{\text{op}})$. Intuitively, every element in $\mathcal{K}(\mathbf{F})(n)$ is a tuple with index set $\bigcup_{m \in \text{Ob}(\mathbf{L}_\Sigma^{\text{op}})} \mathbf{L}_\Sigma^{\text{op}}[n, m]$, and its component at index $\theta: n \rightarrow m$ is an element in $\mathbf{F}(m)$. We follow the convention of [3] and write \dot{x}, \dot{y}, \dots for such tuples, and $\dot{x}(\theta)$ for the component of \dot{x} at index θ .
- With this convention, given an arrow $\sigma \in \mathbf{L}_\Sigma^{\text{op}}[n, n']$, $\mathcal{K}(\mathbf{F})(\sigma)$ is defined by pointwise substitution as the mapping of the tuple \dot{x} to the tuple $\langle \dot{x}(\theta \circ \sigma) \rangle_{\theta: n' \rightarrow m}$.
- On arrows, given a morphism $\alpha: \mathbf{F} \rightarrow \mathbf{G}$ in $\mathbf{Sets}^{|\mathbf{L}_\Sigma^{\text{op}}|}$, $\mathcal{K}(\alpha)$ is a natural transformation $\mathcal{K}(\mathbf{F}) \rightarrow \mathcal{K}(\mathbf{G})$ defined pointwisely as $\mathcal{K}(\alpha)(n): \dot{x} \mapsto \langle \alpha_m(\dot{x}(\theta)) \rangle_{\theta: n \rightarrow m}$.

10:12 A Coalgebraic Perspective on Probabilistic Logic Programming

It is also useful to record the unit $\eta: 1 \rightarrow \mathcal{K}\mathcal{U}$ of the adjunction $\mathcal{U} \dashv \mathcal{K}$. Given a presheaf $F: \mathbf{L}_\Sigma^{\text{op}} \rightarrow \mathbf{Sets}$, $\eta_F: F \rightarrow \mathcal{K}\mathcal{U}F$ is a natural transformation defined by $\eta_F(n): x \mapsto \langle F(\theta)(x) \rangle_{\theta: n \rightarrow m}$.

Saturation in PLP. We now come back to the question of the coalgebra structure on the presheaf At modelling PLP. First, we are now able to represent p in (5) as a coalgebra map. The aforementioned naturality issue is solved by defining it as a morphism in $\mathbf{Sets}^{\mathbf{L}_\Sigma^{\text{op}}}$ rather than in $\mathbf{Sets}^{\mathbf{L}_\Sigma^{\text{op}}}$, thus making naturality trivial. The coalgebra p will have the following type

$$p: \mathcal{U}\text{At} \rightarrow \widehat{\mathcal{M}}_{pr} \widehat{\mathcal{P}}_c \widehat{\mathcal{P}}_f \mathcal{U}\text{At} \quad (7)$$

where $\widehat{(\cdot)}$ is the obvious extension of \mathbf{Sets} -endofunctors to $\mathbf{Sets}^{\mathbf{L}_\Sigma^{\text{op}}}$ -endofunctors, defined by functor precomposition. With respect to the ground case, note the insertion of $\widehat{\mathcal{P}}_c$, the lifting of the *countable* powerset functor, in order to account for the countably many instances of a clause that may match the given goal (*cf.* the discussion below (5)).

► **Example 16.** Our program \mathbb{P}^{al} (Example 14) is based on $\text{At}_{al}: \mathbf{L}_{\Sigma_{al}}^{\text{op}} \rightarrow \mathbf{Sets}$. Some of its values are $\text{At}_{al}(0) = \{\text{Mary}, \text{Neigh}(\text{Mary}), \text{Neigh}(\text{Neigh}(\text{Mary})), \dots\}$ and $\text{At}_{al}(1) = \{x, \text{Mary}, \text{Neigh}(x), \text{Neigh}(\text{Mary}), \dots\}$. Part of the coalgebra p_{al} modelling the program \mathbb{P}^{al} is as follows (*cf.* the tree (4)).

$$\begin{aligned} (p_{al})_0(\text{Hear_alarm}(\text{Mary})) &= 0.8\{\{\text{Alarm}, \text{Wake}(\text{Mary})\}\} + 0.3\{\{\text{Parasusia}(\text{Mary})\}\} \\ (p_{al})_1(\text{Alarm}) &= 0.5\{\{\text{Earthquake}\}\} + 0.9\{\{\text{Burglary}\}\} \\ &\quad + 0.1\{\{\text{PassBy}(\text{Mary})\}, \{\text{PassBy}(\text{Neigh}(\text{Mary}))\}, \{\text{PassBy}(\text{Neigh}(x))\}, \dots\} \end{aligned}$$

The universal property of the adjunction (6) gives a canonical “lifting” of p to a $\mathcal{K}\widehat{\mathcal{M}}_{pr} \widehat{\mathcal{P}}_c \widehat{\mathcal{P}}_f \mathcal{U}$ -coalgebra p^\sharp on At , performing unification rather than just term-matching:

$$p^\sharp := \text{At} \xrightarrow{\eta_{\text{At}}} \mathcal{K}\mathcal{U}\text{At} \xrightarrow{\mathcal{K}p} \mathcal{K}\widehat{\mathcal{M}}_{pr} \widehat{\mathcal{P}}_c \widehat{\mathcal{P}}_f \mathcal{U}\text{At} \quad (8)$$

where η is the unit of the adjunction, as defined above. Spelling it out, p^\sharp is the mapping

$$p_n^\sharp : A \in \text{At}(n) \mapsto \langle p_m(A\theta) \rangle_{\theta: n \rightarrow m}.$$

Intuitively, p_n^\sharp retrieves all the unifiers $\langle \theta, \tau \rangle$ of A and head H in \mathbb{P} : first, we have $A\theta \in \text{At}(m)$ as a component of the saturation of A by η_{At} ; then we term-match H with $A\theta$ by $\mathcal{K}p_m$.

► **Remark 17.** Note that the parameter $n \in \text{Ob}(\mathbf{L}_\Sigma^{\text{op}})$ in the natural transformation p^\sharp fixes the pool $\{x_1, \dots, x_n\}$ of variables appearing in the atoms (and relative substitutions) that are considered in the computation. Analogously to the case of pure logic programs [17, 3], it is intended that such n can always be chosen “big enough” so that all the relevant substitution instances of the current goal and clauses in the program are covered – note the variables occurring therein always form a *finite* set, included in $\{x_1, \dots, x_m\}$ for some $m \in \mathbb{N}$.

4.2 Derivation Semantics

Once we have identified our coalgebra type, the construction leading to the derivation semantics $\llbracket - \rrbracket_{p^\sharp}$ for general PLP is completely analogous to the ground case. One can define the cofree coalgebra for $\mathcal{K}\widehat{\mathcal{M}}_{pr} \widehat{\mathcal{P}}_c \widehat{\mathcal{P}}_f \mathcal{U}(-)$ by terminal sequence, similarly to Construction 7. For simplicity, henceforth we denote the functor $\mathcal{K}\widehat{\mathcal{M}}_{pr} \widehat{\mathcal{P}}_c \widehat{\mathcal{P}}_f \mathcal{U}(-)$ by \mathcal{S} .

► **Construction 18.** The terminal sequence for $\text{At} \times \mathcal{S}(-) : \mathbf{Sets}^{\mathbf{L}_\Sigma^{\text{op}}} \rightarrow \mathbf{Sets}^{\mathbf{L}_\Sigma^{\text{op}}}$ consists of a sequence of objects X_α and morphisms $\delta_\alpha^\beta: X_\beta \rightarrow X_\alpha$, for $\alpha < \beta \in \mathbf{Ord}$, defined analogously to Construction 7, with p^\sharp and \mathcal{S} replacing p and $\mathcal{M}_{pr}\mathcal{P}_f$.

This terminal sequence converges by the following lemma.

► **Proposition 19.** *\mathcal{S} is accessible, and preserves monomorphisms.*

Proof. Since both properties are preserved by composition, it suffices to show that they hold for all the component functors. For $\widehat{\mathcal{M}}_{pr}$, $\widehat{\mathcal{P}}_c$ and $\widehat{\mathcal{P}}_f$, they follow from accessibility and mono-preservation of \mathcal{M}_{pr} , \mathcal{P}_c and \mathcal{P}_f (see Proposition 8), as (co)limits in presheaf categories are computed pointwise. For \mathcal{K} and \mathcal{U} , these properties are proven in [3]. ◀

Therefore the terminal sequence for $\text{At} \times \mathcal{S}(-)$ converges at some limit ordinal, say γ , yielding the final $\text{At} \times \mathcal{S}(-)$ -coalgebra $X_\gamma \xrightarrow{\cong} \text{At} \times \mathcal{S}(X_\gamma)$. The derivation semantics is then defined $\llbracket - \rrbracket_{p^\sharp} : \text{At} \rightarrow X_\gamma$ by universal property, as on the right.

$$\begin{array}{ccc}
 \text{At} & \xrightarrow{\llbracket - \rrbracket_{p^\sharp}} & X_\gamma \\
 \langle \text{id}_{\text{At}}, p^\sharp \rangle \downarrow & & \downarrow \cong \\
 \text{At} \times \mathcal{S}(\text{At}) & \xrightarrow{\text{id}_{\text{At}} \times \llbracket - \rrbracket_{p^\sharp}} & \text{At} \times \mathcal{S}(X_\gamma)
 \end{array} \tag{9}$$

A careful inspection of the terminal sequence constructing X_γ allows to infer a representation of its elements as trees, among which we have those representing computations by unification of goals in a PLP program. We call these *stochastic saturated derivation trees*, as they extend the derivation trees of Definition 5 and are the probabilistic variant of saturated and-or trees in [3]. Using (9) one can easily verify that $\llbracket A \rrbracket$ is indeed the stochastic saturated derivation tree for a given goal A . Example 15 provides a pictorial representation of one such tree.

► **Definition 20** (Stochastic saturated derivation trees). *Given a probabilistic logic program \mathbb{P} , a natural number n and an atom $A \in \text{At}(n)$. The stochastic saturated derivation tree for A in \mathbb{P} is the possibly infinite tree \mathcal{T} satisfying the following properties:*

1. *There are four kinds of nodes: atom-node (labelled with an atom), substitution-node (labelled with a substitution), clause-node (labelled with \bullet), instance-node (labelled with \blacklozenge), appearing alternatively in depth in this order. The root is an atom-node with label A .*
2. *Each clause-node is labelled with a probability value.*
3. *Suppose an atom-node s is labelled with $A' \in \text{At}(n')$. For every substitution $\theta : n' \rightarrow m'$, s has exactly one (substitution-) child t labelled with θ . For every clause $r :: H \leftarrow B_1, \dots, B_k$ in \mathbb{P} such that H matches $A'\theta$ (via some substitution), t has exactly one (clause-) child u , and edge $t \rightarrow u$ is labelled with r . Then for every substitution τ such that $A'\theta = H\tau$ and $B_1\tau, \dots, B_k\tau \in \text{At}(m')$, u has exactly one (instance-) child v . Also v has exactly $|\{B_1\tau, \dots, B_k\tau\}|$ -many (atom-) children, each labelled with one element in $\{B_1\tau, \dots, B_k\tau\}$.*

4.3 Distribution Semantics

In this section we conclude by giving a coalgebraic perspective on the distribution semantics $\langle\langle - \rangle\rangle$ for general PLP. Mimicking the ground case (Section 3.3), this will be presented as an extension of the derivation semantics, via a “possible worlds” natural transformation. Also in the general case, we want to guarantee that a single probability value is computable for a given goal A from the corresponding tree $\langle\langle A \rangle\rangle$ in the final coalgebra – whenever this probability

is also computable in the “traditional” way (see (1)) of giving distribution semantics to PLP. In this respect, the presence of variables and substitutions poses additional challenges, for which we refer to Appendices A and B. In a nutshell, the issue is that the distribution semantics counts the use of a clause in the program at most once, independently from how many times that clause is used again in the computation. To account for this aspect in our tree representation, we need to give enough information to determine which clause is used at each step of the computation, so that a second use can be easily detected. Note that neither our saturated derivation trees, nor a “naive” extension of them to distribution trees, carry such information: what appears in there is only the instantiated heads and bodies, but in general one cannot retrieve A from a substitution θ and the instantiation $A\theta$. This is best illustrated via a simple example.

► **Example 21.** Consider the following program, based on the signature $\Sigma = \{a^0\}$ and two 1-ary predicates P, Q . It consists of two clauses:

$$0.5 :: P(x_1) \leftarrow Q(x_1) \mid 0.5 :: P(x_1) \leftarrow Q(x_2)$$

The goal $P(a)$ matches the head of both clauses. However, given the sole information of the next goal being $Q(a)$, it is impossible to say whether the first clause has been used, instantiated with $x_1 \mapsto a$, or the second clause has been used, instantiated with $x_1 \mapsto a, x_2 \mapsto a$.

This observation motivates, as intermediate step towards the distribution semantics, the addition of labels to clause-nodes in derivation trees, in order to make explicit which clause is being applied. From the coalgebraic viewpoint, this just amounts to an extension of the type of the term-matching coalgebra:

$$\tilde{p}: \mathcal{U}\text{At} \rightarrow \widehat{\mathcal{M}}_{pr}(\widehat{\mathcal{P}}_c \widehat{\mathcal{P}}_f \mathcal{U}\text{At} \times (\mathcal{U}\text{At} \times \mathcal{U}\widehat{\mathcal{P}}_f \text{At})).$$

Note the insertion of $(-) \times (\mathcal{U}\text{At} \times \mathcal{U}\widehat{\mathcal{P}}_f \text{At})$, which allows us to indicate at each step the head ($\mathcal{U}\text{At}$) and the body ($\mathcal{U}\widehat{\mathcal{P}}_f \text{At}$) of the clause being used, its probability label being already given by $\widehat{\mathcal{M}}_{pr}$. More formally, for any n and atom $A \in \text{At}(n)$, we define¹

$$\tilde{p}_n(A): \langle \{B_1\tau_i, \dots, B_k\tau_i\}_{i \in \mathbb{I} \subseteq \mathbb{N}}, \langle H, \{B_1, \dots, B_k\} \rangle \rangle \mapsto \begin{cases} r & (r :: H \leftarrow B_1, \dots, B_k) \in \mathbb{P}, H\tau_i = A \\ 0 & \text{otherwise} \end{cases}$$

As in the case of p in (7), we can move from term-matching to unification by using the universal property of the adjunction $\mathcal{U} \dashv \mathcal{K}$, yielding $\tilde{p}^\sharp: \text{At} \rightarrow \mathcal{K}\widehat{\mathcal{M}}_{pr}(\widehat{\mathcal{P}}_c \widehat{\mathcal{P}}_f \mathcal{U}\text{At} \times (\mathcal{U}\text{At} \times \mathcal{U}\widehat{\mathcal{P}}_f \text{At}))$. For simplicity henceforth we denote the functor $\mathcal{K}\widehat{\mathcal{M}}_{pr}(\widehat{\mathcal{P}}_c \widehat{\mathcal{P}}_f \mathcal{U}(-) \times (\mathcal{U}\text{At} \times \widehat{\mathcal{P}}_f \text{At}))$ by \mathcal{R} .

We are now able to conclude our characterisation of the distribution semantics. The “possible worlds” transformation $\text{pw}: \mathcal{M}_{pr} \Rightarrow \mathcal{D}_{\leq 1} \mathcal{P}_f$ (Definition 13) yields a natural transformation $\widehat{\text{pw}}: \widehat{\mathcal{M}}_{pr} \rightarrow \widehat{\mathcal{D}}_{\leq 1} \widehat{\mathcal{P}}_f$, defined pointwise by pw . We can use $\widehat{\text{pw}}$ to translate \mathcal{R} into the functor $\mathcal{K}\widehat{\mathcal{D}}_{\leq 1} \widehat{\mathcal{P}}_f(\widehat{\mathcal{P}}_c \widehat{\mathcal{P}}_f \mathcal{U}(-) \times (\mathcal{U}\text{At} \times \widehat{\mathcal{P}}_f \text{At}))$, abbreviated as \mathcal{O} , which is going to give the type of saturated distribution trees for general PLP programs.

¹ As noted in Remark 17, instantiating \tilde{p} to some $n \in \text{Ob}(\mathbf{L}_\Sigma^{\text{op}})$ fixes a variable context $\{x_1, \dots, x_n\}$ both for the goal and the clause labels. In practice, because the set of clauses is always finite, it suffices to chose n “big enough” so that the variables appearing in the clauses are included in $\{x_1, \dots, x_n\}$.

As a simple extension of the developments in Section 4.2, we can construct the cofree \mathcal{R} -coalgebra $\Phi \xrightarrow{\cong} \text{At} \times \mathcal{R}(\Phi)$ via a terminal sequence. Similarly, one can obtain the cofree \mathcal{O} -coalgebra $\Psi \xrightarrow{\cong} \text{At} \times \mathcal{O}(\Psi)$. By the universal property of Ψ , all these ingredients get together in the definition of the distribution semantics $\langle\langle - \rangle\rangle_{\tilde{p}^\#}$ for arbitrary PLP programs $\tilde{p}^\#$

$$\begin{array}{ccccc}
 & & \langle\langle - \rangle\rangle_{\tilde{p}^\#} & & \\
 & \text{At} & \xrightarrow{\quad !_\Phi \quad} & \Phi & \xrightarrow{\quad !_\Psi \quad} & \Psi \\
 & \downarrow \langle \text{id}_{\text{At}}, \tilde{p}^\# \rangle & & \downarrow \cong & & \downarrow \cong \\
 \text{At} \times \mathcal{R}\text{At} & \xrightarrow{\text{id}_{\text{At}} \times \mathcal{R}(!_\Phi)} & & \text{At} \times \mathcal{R}\Phi & & \text{At} \times \mathcal{O}\Psi \\
 & \downarrow \text{id}_{\text{At}} \times \mathcal{K}_{\widehat{p}^\#} & & \downarrow \text{id}_{\text{At}} \times \mathcal{K}_{\widehat{p}^\#} & & \\
 \text{At} \times \mathcal{O}\text{At} & \xrightarrow{\text{id}_{\text{At}} \times \mathcal{O}(!_\Phi)} & & \text{At} \times \mathcal{O}\Phi & \xrightarrow{\text{id}_{\text{At}} \times \mathcal{O}(!_\Psi)} & \text{At} \times \mathcal{O}\Psi
 \end{array}$$

where $!_\Phi$ and $!_\Psi$ are given by the evident universal properties, and show the role of the cofree \mathcal{R} -coalgebra Φ as an intermediate step. The layered construction of final coalgebras Ψ and Φ , together with the above characterisation of $\langle\langle - \rangle\rangle_{\tilde{p}^\#}$, allow to conclude that the distribution semantics for the program $\tilde{p}^\#$ maps a goal A to its *saturated distribution tree* $\langle\langle A \rangle\rangle_{\tilde{p}^\#}$, as formally defined below.

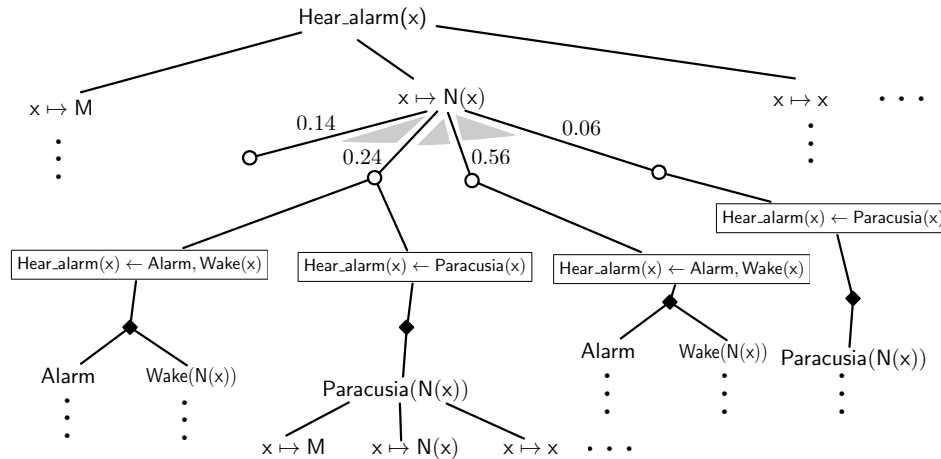
► **Definition 22** (Saturated distribution tree). *The saturated distribution tree for $A \in \text{At}(n)$ in \mathbb{P} is the possibly infinite \mathcal{T} satisfying the following properties based on Definition 20:*

1. *There are five kinds of nodes: in addition to the atom-, substitution-, clause- and instance-nodes, there are world-nodes. The world-nodes are children of the substitution-nodes, and parents of the clause nodes. The root and the order of the rest nodes are the same as in Definition 20, condition 1. The clause-nodes are now labelled with clauses of \mathbb{P} .*
2. *Suppose s is an atom node labelled with $A' \in \text{At}(n')$, and t is a substitution-child of s labelled with $\theta: n' \rightarrow m$. Let C be the set of all clauses \mathcal{C} such that $\text{Head}(\mathcal{C})$ matches $A'\theta$. Then t has $2^{|C|}$ world-children, each representing a subset X of C . If a child u represents subset X , then the edge $t \rightarrow u$ has probability label $\prod_{\mathcal{C} \in X} \text{Label}(\mathcal{C}) \cdot \prod_{\mathcal{C}' \in C \setminus X} \text{Label}(\mathcal{C}')$. Also u has $|X|$ clause-children, one for each clause $\mathcal{C} \in X$, labelled with the corresponding clause. The rest for clause-nodes and instance-nodes are the same as in Definition 20, condition 3.*

► **Remark 23.** Note that, in principle, saturated distribution trees could be defined coalgebraically without the intermediate step of adding clause labels. This is to be expected: coalgebra typically captures the one-step, “local” behaviour of a system. On the other hand, as explained, the need for clause labels is dictated by a computational aspect involving the depth of distribution trees, that is, a “non-local” dimension of the system.

We conclude with the pictorial representation of the saturated distribution tree of a goal in our leading example.

► **Example 24.** In the context of Example 14, the tree $\llbracket \text{Hear_alarm}(x) \rrbracket$ capturing the distribution semantics of $\text{Hear_alarm}(x)$ is (partially) depicted as follows. Note the presence of clauses labelling the clause-nodes.



References

- 1 Falk Bartels, Ana Sokolova, and Erik P. de Vink. A hierarchy of probabilistic system types. *Theor. Comput. Sci.*, 327(1-2):3–22, 2004. doi:10.1016/j.tcs.2004.07.019.
- 2 Filippo Bonchi and Fabio Zanasi. Saturated semantics for coalgebraic logic programming. In *Algebra and Coalgebra in Computer Science - 5th International Conference, CALCO 2013, Warsaw, Poland, September 3-6, 2013. Proceedings*, pages 80–94, 2013. doi:10.1007/978-3-642-40206-7_8.
- 3 Filippo Bonchi and Fabio Zanasi. Bialgebraic semantics for logic programming. *Logical Methods in Computer Science*, 11(1), 2015. doi:10.2168/LMCS-11(1:14)2015.
- 4 Fredrik Dahlqvist, Vincent Danos, Ilias Garnier, and Ohad Kammar. Bayesian inversion by ω -complete cone duality. In *27th International Conference on Concurrency Theory, CONCUR 2016, August 23-26, 2016, Québec City, Canada*, pages 1:1–1:15, 2016. doi:10.4230/LIPIcs.CONCUR.2016.1.
- 5 Eugene Dantsin. Probabilistic logic programs and their semantics. In A. Voronkov, editor, *Logic Programming*, pages 152–164, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.
- 6 Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI'07*, pages 2468–2473, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc. URL: <http://dl.acm.org/citation.cfm?id=1625275.1625673>.
- 7 Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI'07*, pages 2468–2473, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc. URL: <http://dl.acm.org/citation.cfm?id=1625275.1625673>.
- 8 Didier Dubois, Lluas Godo, and Henri Prade. Weighted logics for artificial intelligence : an introductory discussion. *International Journal of Approximate Reasoning*, 55(9):1819–1829, 2014. Weighted Logics for Artificial Intelligence. doi:10.1016/j.ijar.2014.08.002.
- 9 Gopal Gupta, Ajay Bansal, Richard Min, Luke Simon, and Ajay Mallya. Coinductive logic programming and its applications. In Véronica Dahl and Ilkka Niemelä, editors, *Logic Programming*, pages 27–44, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
- 10 Gopal Gupta and Vítor Santos Costa. Optimal implementation of and-or parallel prolog. *Future Generation Computer Systems*, 10(1):71–92, 1994. PARLE '92. doi:10.1016/0167-739X(94)90052-3.

- 11 Bart Jacobs, Aleks Kissinger, and Fabio Zanasi. Causal inference by string diagram surgery. In *Foundations of Software Science and Computation Structures - 22nd International Conference, FOSSACS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Proceedings*, 2019. URL: <http://arxiv.org/abs/1811.08338>.
- 12 Bart Jacobs and Fabio Zanasi. The logical essentials of bayesian reasoning. In Joost-Peter Katoen Gilles Barthe and Alexandra Silva, editors, *Probabilistic Programming*. Cambridge University Press, Cambridge, 2019. URL: <http://arxiv.org/abs/1804.01193>.
- 13 Kristian Kersting and Luc De Raedt. Bayesian logic programming: Theory and tool. In *Introduction to Statistical Relational Learning*, pages 291–322. MIT Press; Cambridge, 2007. URL: <https://lirias.kuleuven.be/retrieve/86539>.
- 14 Ekaterina Komendantskaya and Yue Li. Productive corecursion in logic programming. *TPLP*, 17(5-6):906–923, 2017. doi:10.1017/S147106841700028X.
- 15 Ekaterina Komendantskaya and Yue Li. Towards coinductive theory exploration in horn clause logic: Position paper. In *Proceedings 5th Workshop on Horn Clauses for Verification and Synthesis, HCVS 2018, Oxford, UK, 13th July 2018.*, pages 27–33, 2018. doi:10.4204/EPTCS.278.5.
- 16 Ekaterina Komendantskaya, Guy McCusker, and John Power. Coalgebraic semantics for parallel derivation strategies in logic programming. In *Algebraic Methodology and Software Technology - 13th International Conference, AMAST 2010, Lac-Beauport, QC, Canada, June 23-25, 2010. Revised Selected Papers*, pages 111–127, 2010. doi:10.1007/978-3-642-17796-5_7.
- 17 Ekaterina Komendantskaya and John Power. Coalgebraic semantics for derivations in logic programming. In *Algebra and Coalgebra in Computer Science - 4th International Conference, CALCO 2011, Winchester, UK, August 30 - September 2, 2011. Proceedings*, pages 268–282, 2011. doi:10.1007/978-3-642-22944-2_19.
- 18 Ekaterina Komendantskaya and John Power. Logic programming: Laxness and saturation. *J. Log. Algebr. Meth. Program.*, 101:1–21, 2018. doi:10.1016/j.jlamp.2018.07.004.
- 19 Ekaterina Komendantskaya, John Power, and Martin Schmidt. Coalgebraic logic programming: from semantics to implementation. *J. Log. Comput.*, 26(2):745–783, 2016. doi:10.1093/logcom/exu026.
- 20 John W. Lloyd. *Foundations of Logic Programming, 2nd Edition*. Springer, 1987. doi:10.1007/978-3-642-83189-8.
- 21 Saunders MacLane. *Categories for the Working Mathematician*. Springer-Verlag, 1971. Graduate Texts in Mathematics, Vol. 5.
- 22 Søren Mørk and Ian Holmes. Evaluating bacterial gene-finding hmm structures as probabilistic logic programs. *Bioinformatics*, 28(5):636–642, 2012. doi:10.1093/bioinformatics/btr698.
- 23 Raymond Ng and V.S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, 1992. doi:10.1016/0890-5401(92)90061-J.
- 24 Fabrizio Riguzzi and Terrance Swift. Probabilistic logic programming under the distribution semantics, 2014.
- 25 Taisuke Sato. A statistical learning method for logic programs with distribution semantics. In *Logic Programming, Proceedings of the Twelfth International Conference on Logic Programming, Tokyo, Japan, June 13-16, 1995*, pages 715–729, 1995.
- 26 Alexandra Silva. Kleene coalgebra. Phd thesis, CWI, Amsterdam, The Netherlands, 2010.
- 27 Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT Press, Cambridge, Mass., 2005.
- 28 James Worrell. Terminal sequences for accessible endofunctors. *Electronic Notes in Theoretical Computer Science*, 19:24–38, 1999. CMCS’99, Coalgebraic Methods in Computer Science. doi:10.1016/S1571-0661(05)80267-1.
- 29 Riccardo Zese. *Probabilistic Semantic Web: Reasoning and Learning*. IOS Press, Amsterdam, The Netherlands, The Netherlands, 2017.

A

 Computability of the Distribution Semantics (Ground Case)

Computing with distribution trees. As a justification for our tree representation of the distribution semantics, we claimed that the probability $\Pr_{\mathbb{P}}(A)$ associated with a goal (see (1)) can be straightforwardly computed from the corresponding distribution tree $\langle\langle A \rangle\rangle_{\mathbb{P}}$. This appendix supplies such an algorithm. Note this serves just as a proof of concept, without any claim of efficiency compared to pre-existing implementations. In the sequel we fix a ground PLP program \mathbb{P} with atoms At , a goal $A \in \text{At}$ and the distribution tree \mathcal{T} for A in \mathbb{P} (Definition 9). First, we may assume that \mathcal{T} does not contain loop (which implies that \mathcal{T} is finite). Indeed, in the ground case loops only results from multiple appearance of an atom in some path, which can be easily detected. We can prune the subtrees of \mathcal{T} rooted by atoms that already appeared at an earlier stage: this does not affect the computation of $\Pr_{\mathbb{P}}(A)$, and it makes \mathcal{T} finite. Next, we introduce the concept of *deterministic* subtree. Basically a deterministic subtree selects one world-node at each stage. Recall that every clause-node in \mathcal{T} represents a clause in $|\mathbb{P}|$, whose head is the label of its atom-grandparent, and body consists of the labels of its atom-children.

► **Definition 25.** A subtree \mathcal{S} of \mathcal{T} is deterministic if (i) it contains exactly one child (world-node) for each atom-node and all children for other nodes, and (ii) for any distinct atom-nodes s, t in \mathcal{S} with the same label, s and t have their clause-grandchildren representing the same clauses.

The idea is that \mathcal{S} describes a computation in which the choice of a possible world (i.e., a sub-program of \mathbb{P}) associated to any atom B appearing during the resolution is uniquely determined. Because of this feature, each deterministic subtree uniquely identifies a set of sub-programs of \mathbb{P} , and together the deterministic subtrees of \mathcal{T} form a *partition* over the set of these sub-programs (see Proposition 27 below).

Since \mathcal{T} is finite, it is clear that we can always provide an enumeration of its deterministic subtrees. We can now present our algorithm, in two steps. First, Algorithm 1 computes the probability associated with a deterministic subtree. Second, Algorithm 2 computes $\Pr_{\mathbb{P}}(A)$ by summing up the probabilities found by Algorithm 1 on all the deterministic subtrees of \mathcal{T} which contains a refutation of A . Below we write $\text{label}(s \rightarrow t)$ for the probability labelling the edge from s to t .

■ **Algorithm 1** Compute probability of a deterministic subtree.

Input: A deterministic subtree \mathcal{S} of \mathcal{T}
Output: The probability of \mathcal{S}

```

1: probList = [ ]
2: for atom-node  $s$  in  $\mathcal{S}$  do
3:   if  $s$  has child then
4:     probList += label( $s \rightarrow \text{child}(s)$ )
5: if probList == [ ] then
6:   return 0
7: else prob = product of values in probList
8:   return prob

```

■ **Algorithm 2** Compute probability of a goal.

Input: The distribution tree \mathcal{T} of A in \mathbb{P}

Output: The success probability $\Pr_{\mathbb{P}}(A)$

```

1: probSuc = 0
2: for deterministic subtree  $\mathcal{S}$  of  $\mathcal{T}$  do
3:   if  $\mathcal{S}$  refutes  $A$  then
4:     probSuc += Algorithm 1( $\mathcal{S}$ )
5: return probSuc

```

The above procedure terminates because \mathcal{T} is finite and every for-loop is finite. We now focus on the correctness of the algorithm.

Correctness. As mentioned, a world-node in a deterministic subtree can be seen as a choice of clauses: one chooses the clauses represented by its clause-children, and discards the clauses represented by its “complement” world. For correctness, we make this precise, via the following definition.

► **Definition 26.** *Given a clause \mathcal{C} in \mathbb{P} , a deterministic subtree \mathcal{S} of \mathcal{T} , a world-node t and its atom-parent s in \mathcal{S} , we say t accepts \mathcal{C} if $\text{Head}(\mathcal{C}) = \text{label}(s)$ and there is a clause-child of t that represents \mathcal{C} ; t rejects \mathcal{C} if $\text{Head}(\mathcal{C}) = \text{label}(s)$ but no clause-child of t represents \mathcal{C} . We say \mathcal{S} accepts (rejects) \mathcal{C} if there exists a world-node t in \mathcal{S} accepts (rejects) \mathcal{C} .*

Note that Definition 25, condition (ii) prevents the existence of world-nodes t, t' in \mathcal{S} such that t accepts \mathcal{C} and t' rejects \mathcal{C} . Thus the notion that \mathcal{S} accepts (rejects) \mathcal{C} is well-defined. We denote the set of clauses accepted and rejected by \mathcal{S} by $\text{Acc}(\mathcal{S})$ and $\text{Rej}(\mathcal{S})$, respectively. Then we can define the set $\text{SubProg}(\mathcal{S})$ of sub-programs represented by \mathcal{S} as

$$\text{SubProg}(\mathcal{S}) := \{\mathbb{L} \subseteq |\mathbb{P}| \mid \forall \mathcal{C} \in \text{Acc}(\mathcal{S}), \mathcal{C} \in \mathbb{L}; \forall \mathcal{C}' \in \text{Rej}(\mathcal{S}), \mathcal{C}' \notin \mathbb{L}\} \quad (10)$$

We will prove the correctness of the algorithm through the following basic observations on the connection between deterministic subtrees and the sub-programs they represent:

► **Proposition 27.** *Suppose \mathcal{S} is a deterministic subtree of the distribution tree \mathcal{T} of A .*

1. $\{\text{SubProg}(\mathcal{S}) \mid \mathcal{S} \text{ is deterministic subtree of } \mathcal{T}\}$ forms a partition of $\mathcal{P}(\mathbb{P})$.
2. Either $\mathbb{L} \vdash A$ for all $\mathbb{L} \in \text{SubProg}(\mathcal{S})$ or $\mathbb{L} \not\vdash A$ for all $\mathbb{L} \in \text{SubProg}(\mathcal{S})$.
3. $\sum_{\mathbb{L} \in \text{SubProg}(\mathcal{S})} \Pr_{\mathbb{P}}(\mathbb{L}) = \prod_{r_i \in \mathcal{S}} r_i$, where the r_i s are all the probability labels appearing in \mathcal{S} (on the atom-node \rightarrow world-node edges).

Proof.

1. Given any two distinct deterministic subtrees, there is an atom-node s such that the subtrees include distinct world-child of s . So by (10) the sub-programs they represent do not share at least one clause. Moreover, given a sub-program \mathbb{L} , one can always identify a deterministic subtree \mathcal{S} such that $\mathbb{L} \in \text{SubProg}(\mathcal{S})$, as follows: given the A -labelled root of \mathcal{T} , select the world-child w of A representing the (possibly empty) set X of all clauses in \mathbb{L} whose head is A ; then select the children (if any) of w , and repeat the procedure.
2. Note that a sub-program $\mathbb{L} \in \text{SubProg}(\mathcal{S})$ refutes the goal A iff \mathcal{S} contains a successful refutation of A , and the latter property is independent of the choice of \mathbb{L} .

10:20 A Coalgebraic Perspective on Probabilistic Logic Programming

3. We refer to $\prod_{r_i \in \mathcal{S}} r_i$ as the probability of the deterministic subtree \mathcal{S} . For each sub-program $\mathbb{L} \in \text{SubProg}(\mathcal{S})$, its probability can be written as

$$\Pr_{\mathbb{P}}(\mathbb{L}) = \prod_{\mathcal{C} \in \text{Acc}(\mathcal{S})} \text{Label}(\mathcal{C}) \cdot \prod_{\mathcal{C}' \in \text{Rej}(\mathcal{S})} (1 - \text{Label}(\mathcal{C}')) \cdot \Pr_{\mathbb{P} \setminus (\text{Acc} \cup \text{Rej})}(\mathbb{L} \setminus \text{Acc}(\mathcal{S})) \quad (11)$$

Note that $\text{SubProg}(\mathcal{S})$ can also be written as $\{X \cup \text{Acc}(\mathcal{S}) \mid X \subseteq \mathbb{P} \setminus (\text{Acc}(\mathcal{S}) \cup \text{Rej}(\mathcal{S}))\}$, so

$$\sum_{\mathbb{L} \in \text{SubProg}(\mathcal{S})} \Pr_{\mathbb{P} \setminus (\text{Acc} \cup \text{Rej})}(\mathbb{L} \setminus \text{Acc}(\mathcal{S})) = 1. \quad (12)$$

Applying equation (12) to the sum of (11) over all $\mathbb{L} \in \text{SubProg}(\mathcal{S})$, we get

$$\sum_{\mathbb{L} \in \text{SubProg}(\mathcal{S})} \Pr_{\mathbb{P}}(\mathbb{L}) = \prod_{\mathcal{C} \in \text{Acc}(\mathcal{S})} \text{Label}(\mathcal{C}) \cdot \prod_{\mathcal{C}' \in \text{Rej}(\mathcal{S})} (1 - \text{Label}(\mathcal{C}')) \quad (13)$$

For each world-node t and its atom-parent s , we can use the terminology in Definition 26, and express $\text{label}(s \rightarrow t)$ (see Definition 9) as

$$\text{label}(s \rightarrow t) = \prod_{t \text{ accepts } \mathcal{C}} \text{Label}(\mathcal{C}) \cdot \prod_{t \text{ rejects } \mathcal{C}'} (1 - \text{Label}(\mathcal{C}')). \quad (14)$$

Applying (14) to the whole deterministic subtree \mathcal{S} , we obtain

$$\begin{aligned} \sum_{\mathbb{L} \in \text{SubProg}(\mathcal{S})} \Pr_{\mathbb{P}}(\mathbb{L}) &\stackrel{(13)}{=} \prod_{\mathcal{C} \in \text{Acc}(\mathcal{S})} \text{Label}(\mathcal{C}) \cdot \prod_{\mathcal{C}' \in \text{Rej}(\mathcal{S})} (1 - \text{Label}(\mathcal{C}')) \\ &\stackrel{\text{Def.26}}{=} \prod_{(\text{world-node } t \text{ in } \mathcal{S})} \left[\prod_{t \text{ accepts } \mathcal{C}} \text{Label}(\mathcal{C}) \cdot \prod_{t \text{ rejects } \mathcal{C}'} (1 - \text{Label}(\mathcal{C}')) \right] \\ &\stackrel{(14)}{=} \prod_{r_i \in \mathcal{S}} r_i \end{aligned}$$

If we say two world-nodes t and t' are equivalent if their clause-children represent exactly the same clauses in \mathbb{P} , then the $\prod_{(\text{world-node } t \text{ in } \mathcal{S})}$ in the above calculation visits every world-node exactly once modulo equivalence. \blacktriangleleft

We can now formulate the success probability of A as follows

$$\begin{aligned} \Pr_{\mathbb{P}}(A) &= \sum_{|\mathbb{P}| \supseteq \mathbb{L} \vdash A} \Pr_{\mathbb{P}}(\mathbb{L}) \stackrel{(\text{Prop.27,1\&2})}{=} \sum_{\mathcal{S} \vdash A} \sum_{\mathbb{L} \in \text{SubProg}(\mathcal{S})} \Pr_{\mathbb{P}}(\mathbb{L}) \\ &\stackrel{(13)}{=} \sum_{\mathcal{S} \vdash A} \left[\prod_{\mathcal{C} \in \text{Acc}(\mathcal{S})} \text{Label}(\mathcal{C}) \cdot \prod_{\mathcal{C}' \in \text{Rej}(\mathcal{S})} (1 - \text{Label}(\mathcal{C}')) \right] \stackrel{(\text{Prop.27,3})}{=} \sum_{\mathcal{S} \vdash A} \prod_{r_i \in \mathcal{S}} r_i \end{aligned}$$

In words, this is exactly Algorithm 2: we sum up the probabilities of all deterministic subtrees \mathcal{S} of the distribution tree \mathcal{T} which contain a proof of A .

B Computability of the Distribution Semantics (General Case)

Computability of the distribution semantics for arbitrary PLP programs relies on the substitution mechanism employed in the resolution. This aspect deserves a preliminary discussion. Traditionally, logic programming has both the theorem-proving and problem-solving perspectives [18]. From the problem-solving perspective, the aim is to find a refutation of the goal $\leftarrow G$, which amounts to finding a proof of *some substitution instance* of G . From

the theorem-proving perspective, the aim is to search for a proof of the goal G itself as an atom. The main difference is in the substitution mechanism of resolution: unification for the problem-solving and term-matching for the theorem-proving perspective. We will first explore computability within the theorem-proving perspective. As resolution therein is by term-matching, the probability $\text{Pr}_{\mathbb{P}}^{\text{TM}}(A)$ of proving a goal A in a PLP program \mathbb{P} is formulated as $\text{Pr}_{\mathbb{P}}^{\text{TM}}(A) := \sum_{|\mathbb{P}| \supseteq \mathbb{L} \Rightarrow A} \text{Pr}_{\mathbb{P}}(\mathbb{L})$, where $\mathbb{L} \Rightarrow A$ means that A is derivable in the sub-program \mathbb{L} (not to be confused with $\mathbb{L} \vdash A$, which stands for *some substitution instance* of A being derivable in \mathbb{L} , see (1)).

In our coalgebraic framework, the distribution semantics for general PLP programs is represented on “saturated” trees, in which computations are performed by unification. However, following [3], one can define the *TM (Term Matching) distribution tree* of a goal A in a program \mathbb{P} by “desaturation” of the saturated distribution tree for A in \mathbb{P} . The coalgebraic definition, for which we refer to [3], applies pointwise on the saturated tree the counit $\epsilon_{\mathcal{U}\text{At}}: \mathcal{U}\mathcal{K}\mathcal{U}\text{At} \rightarrow \mathcal{U}\text{At}$ of the adjunction $\mathcal{U} \dashv \mathcal{K}$ (cf. (6)). The TM distribution tree which results from “desaturation” can be described very simply: at each layer of the starting saturated distribution tree, one prunes all the subtrees which are not labelled with the identity substitution $\text{id} := x_1 \mapsto x_1, x_2 \mapsto x_2, \dots$. In this way, the only remaining computation are those in which resolution only applies a non-trivial substitution on the clause side, that is, in which unification is restricted to term-matching.

Computability of term-matching distribution semantics. One may compute the success probability $\text{Pr}_{\mathbb{P}}^{\text{TM}}(A)$ in \mathbb{P} from the TM distribution tree of A in \mathbb{P} . The computation goes similarly to Algorithm 2: the problem amounts to calculating the probabilities of those deterministic subtrees of the distribution tree which prove the goal. We confine ourselves to some remarks on the aspects that require extra care, compared to the ground case.

1. The probability $\text{Pr}_{\mathbb{P}}^{\text{TM}}(A)$ is not computable in whole generality. It depends on whether one can decide all the proofs of A in the pure logic program $|\mathbb{P}|$, and there are various heuristics in logic programming for this task.
2. It is still possible to decide whether a subtree is deterministic, but the algorithm in the general case is a bit subtler, as it is now possible that two different goals match the same clause (instantiated in two different ways).
3. When calculating the probability of a deterministic subtree in the TM distribution tree, multiple appearances of a single clause (possibly instantiated with different substitutions) should be counted only once. In order to ensure this one needs to be able to identify which clause is applied at each step of the computation described by the distribution trees: this is precisely the reason of the addition of the clause labels in the coalgebra type of these trees, as discussed in Section 4.3.

We conclude by briefly discussing the problem-solving perspective, in which resolution is based on arbitrary unification rather than just term-matching. In standard SLD-resolution, computability relies on the possibility of identifying the *most general* unifier between a goal and the head of a given clause. This can be done also within saturated distribution trees, since saturation supplies *all* the unifiers, thus in particular the most general one. This means that, on principle, one may compute the distribution semantics based on most general unification from the saturated distributed tree associated with a goal, with similar caveats as the ones we described for the term-matching case. However, the lack of a satisfactory coalgebraic treatment of most general unifiers [3] makes us privilege the theorem-proving perspective discussed above, for which desaturation provides an elegant categorical formalisation. This is also in line with the series of works [17, 19] on coalgebraic (pure) logic programming, all based on term-matching as substitution mechanism.

Sequencing and Intermediate Acceptance: Axiomatisation and Decidability of Bisimilarity

Astrid Belder

Eindhoven University of Technology, Eindhoven, The Netherlands

Bas Luttik

Eindhoven University of Technology, Eindhoven, The Netherlands

Jos Baeten

CWI, Amsterdam, The Netherlands

University of Amsterdam, Amsterdam, The Netherlands

Abstract

The Theory of Sequential Processes includes deadlock, successful termination, action prefixing, alternative and sequential composition. Intermediate acceptance, which is important for the integration of classical automata theory, can be expressed through a combination of alternative composition and successful termination. Recently, it was argued that complications arising from the interplay between intermediate acceptance and sequential composition can be eliminated by replacing sequential composition by sequencing. In this paper we study the equational theory of the recursion-free fragment of the resulting process theory modulo bisimilarity, proving that it is not finitely based, but does afford a ground-complete axiomatisation if a unary auxiliary operator is added. Furthermore, we prove that bisimilarity is decidable for processes definable by means of a finite guarded recursive specification over the process theory.

2012 ACM Subject Classification Theory of computation → Process calculi

Keywords and phrases Sequencing, Sequential composition, Bisimilarity, Axiomatisation, Decidability

Digital Object Identifier 10.4230/LIPIcs.CALCO.2019.11

Acknowledgements We thank the anonymous reviewers for their elaborate reviews.

1 Introduction

Successful termination has been a source of controversy from the early days of process algebra. The process theory CCS [18] does not make the distinction between deadlock and successful termination at all. The process theory ACP [9] does make the distinction semantically, but, although it includes a constant denoting deadlock, it does not, in its original formulation, include a constant denoting the successfully terminated process. Only later proposals were made for including such a constant [1, 24].

From a concurrency-theoretic perspective, including a constant **1** for successful termination raises philosophical questions without clear-cut answers. For instance, what is the behaviour of a process $a.1 + 1$ that may non-deterministically choose between performing the action a and successfully terminating? Can it perform the action a at all? Is it successfully terminated even when it can still perform activity? And what does it mean to sequentially compose $a.1 + 1$ with the process $b.1$? Can $(a.1 + 1) \cdot b.1$ do a b immediately or should it wait until $a.1 + 1$ has performed the a ?

In the classical theory of automata and formal languages, the constant **1** has a more accepted status. The algebras of regular expressions and μ -regular expressions include a constant **1** denoting the language consisting of the empty string. Without the inclusion of the constant, the correspondence between regular expressions and finite automata [16], and the correspondence between μ -regular expressions and pushdown automata [17, 21] would be



© Astrid Belder, Bas Luttik, and Jos Baeten;

licensed under Creative Commons License CC-BY

8th Conference on Algebra and Coalgebra in Computer Science (CALCO 2019).

Editors: Markus Roggenbach and Ana Sokolova; Article No. 11; pp. 11:1–11:22

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

lost. Finite automata and pushdown automata are endowed with an acceptance predicate separate from the transition relation defined on states, and hence they admit *intermediate acceptance*: states may at the same time satisfy the acceptance predicate and have outgoing transitions.

The research presented in this paper is part of a larger project in which we are trying to explore and strengthen connections between the classical theory of automata and formal languages and concurrency theory [3, 4, 5, 6], with the aim to establish a unified theory. Our aim for such a unified theory motivates us to study process algebras including a constant for successful termination.

The operational semantics for sequential composition in the presence of a constant $\mathbf{1}$ denoting successful termination (see, e.g., [2]) prescribes that the sequential composition $(a.\mathbf{1} + \mathbf{1}) \cdot b.\mathbf{1}$ may perform the b transition immediately, on grounds that $a.\mathbf{1} + \mathbf{1}$ satisfies the termination predicate. We refer to this phenomenon as *transparency*. In the presence of recursion, transparency leads to considerable expressiveness; for instance, it facilitates the specification of unboundedly branching behaviour (cf. Example 1 below). Recently, we proposed a revised operational semantics for sequential composition that leads to a different interplay between successful termination and sequential composition [7]. The revised operational rules closely resembles the rules of the *sequencing operator* proposed by Bloom [10], although his theory does not distinguish between deadlock and successful termination. We shall, in this paper, reserve *sequential composition* (denoted by \cdot) for the operator with the operational semantics as described in [2] and use *sequencing* (denoted by $;$) for the operator with the revised operational semantics.

Under the sequencing interpretation, the process $(a.\mathbf{1} + \mathbf{1}) ; b.\mathbf{1}$ cannot perform the b -transition immediately (no transparency); first, the left argument of the sequencing operator must execute until no further activity is possible. The effect of replacing sequential composition by sequencing indirectly changes the interpretation of the constant $\mathbf{1}$: it no longer refers to the option to terminate, but rather signals acceptance. For instance, the process $(a.\mathbf{1} + \mathbf{1}) ; (b.\mathbf{1} + \mathbf{1})$ is in an accepting state since both $a.\mathbf{1} + \mathbf{1}$ and $b.\mathbf{1} + \mathbf{1}$ are accepting; the process $a.\mathbf{1} ; (b.\mathbf{1} + \mathbf{1})$ on the other hand is not in an accepting state.

Replacing sequential composition by sequencing has advantages and disadvantages for the integration of automata theory and concurrency theory. A disadvantage is that language equivalence is not a congruence for sequencing (see Remark 5 at the end of Section 2). As was shown in [7], advantages are that, in the theory with sequencing every context-free behaviour can be simulated by a pushdown automaton up to strong bisimilarity, while this is not the case in the theory with sequential composition, and that every executable processes can be specified, up to divergence-preserving branching bisimilarity, in a process theory without recursion but with a first-order recursive nesting operation.

In this paper, we continue the investigation of the theory of sequential processes with sequencing instead of sequential composition.

First, we consider the equational theory of the recursion-free fragment modulo bisimilarity. We prove that the equational theory is not finitely based (i.e., does not admit a finite equational axiomatisation). Then, we introduce an auxiliary unary operator and prove that, using this auxiliary operator the *ground* equational theory (i.e., the set of all valid equations without variables) admits a finite axiomatisation. And finally we present arguments for the conjecture that, even with the auxiliary operator, the full equational theory (i.e., the set of all valid equations with variables) is not finitely based.

Then, we prove that bisimilarity is decidable for processes definable by means of a guarded recursive specification in the theory with sequencing. To this end, we consider the seminal proof by Christensen, Hüttel and Stirling that bisimilarity is decidable for the theory of

sequential processes without intermediate acceptance [13], and observe that several crucial properties needed in their argument fail in a setting with intermediate acceptance. Our contribution is then to show that, when a form of redundant intermediate acceptance is eliminated from recursive specifications, then these properties are restored and the proof ideas of [13] apply to establish decidability.

This paper is organised as follows: In Section 2 we introduce the Theory of Sequential Processes with sequential composition replaced by sequencing, illustrating the difference between the two operators with an example. In Section 3, we consider the equational theory of the recursion-free fragment. In Section 4, we establish decidability of bisimilarity for processes definable by means of a guarded recursive specification over the Theory of Sequential Processes with sequencing instead of sequential composition. In Section 5 we present some conclusions. For elaborate proofs of the results claimed in this article we refer to the first author’s MSc thesis [8].

2 Sequential Processes

In this section we present the Theory of Sequential Processes adopting the revised operational semantics for sequential composition proposed in [7]. To emphasise that the operational semantics for sequential composition deviates from that in [2], we shall refer to it by the term sequencing and denote it by $;$ instead of by \cdot , reserving \cdot for the variant of sequential composition in [2].

Let \mathcal{A} be a set of *actions*, symbols denoting atomic events, and let \mathcal{P} be a finite set of *process identifiers*. The sets \mathcal{A} and \mathcal{P} serve as parameter of the process theory $\text{TSP}^i(\mathcal{A}, \mathcal{P})$ that we shall introduce below. The set of *process expressions* associated with $\text{TSP}^i(\mathcal{A}, \mathcal{P})$ is generated by the following grammar ($a \in \mathcal{A}$, $X \in \mathcal{P}$):

$$p ::= \mathbf{0} \mid \mathbf{1} \mid a.p \mid p + p \mid p ; p \mid X .$$

The constants $\mathbf{0}$ and $\mathbf{1}$ respectively denote the *deadlocked* (i.e., inactive but not successfully terminated) process and the *successfully terminated* process. For each $a \in \mathcal{A}$ there is a unary action prefix operator $a._$. The binary operators $+$ and $;$ denote alternative composition and sequencing, respectively. We adopt the convention that $a._$ binds strongest and $+$ binds weakest. For a (possibly empty) sequence p_1, \dots, p_n we inductively define $\sum_{i=1}^n p_i = \mathbf{0}$ if $n = 0$ and $\sum_{i=1}^n p_i = (\sum_{i=1}^{n-1} p_i) + p_n$ if $n > 0$. The symbol $;$ is often omitted when writing process expressions. In particular, if $\alpha \in \mathcal{P}^*$, say $\alpha = X_1 \cdots X_n$, then α denotes the process expression inductively defined by $\alpha = \mathbf{1}$ if $n = 0$ and $\alpha = (X_1 \cdots X_{n-1}) ; X_n$ if $n > 0$. We denote by $|\alpha|$ the length of the sequence.

A recursive specification over $\text{TSP}^i(\mathcal{A}, \mathcal{P})$ is a mapping Δ from \mathcal{P} to the set of process expressions associated with $\text{TSP}^i(\mathcal{A}, \mathcal{P})$. The idea is that the process expression p associated with a process identifier $X \in \mathcal{P}$ by Δ *defines* the behaviour of X . We prefer to think of Δ as a collection of *defining equations* $X \stackrel{\text{def}}{=} p$, exactly one for every $X \in \mathcal{P}$. We shall, throughout the paper, presuppose a recursive specification Δ defining the process identifiers in \mathcal{P} , and we shall usually simply write $X \stackrel{\text{def}}{=} p$ for $\Delta(X) = p$. Note that, by our assumption that \mathcal{P} is finite, Δ is finite too.

We associate behaviour with process expressions by defining, on the set of process expressions, a unary acceptance predicate \downarrow (written postfix) and, for every $a \in \mathcal{A}$, a binary transition relation \xrightarrow{a} (written infix), by means of the transition system specification presented in Fig. 1. We write $p \not\xrightarrow{a}$ for “there does not exist p' such that $p \xrightarrow{a} p'$ ” and $p \not\rightarrow$ for “ $p \not\xrightarrow{a}$ for all $a \in \mathcal{A}$ ”. Furthermore, when $w \in \mathcal{A}^*$, say $w = a_1 \dots a_n$, then we write $p \xrightarrow{w} p'$

11:4 Sequencing and Intermediate Acceptance

$$\begin{array}{c}
\frac{}{a.p \xrightarrow{a} p} \quad \frac{p \xrightarrow{a} p'}{p+q \xrightarrow{a} p'} \quad \frac{q \xrightarrow{a} q'}{p+q \xrightarrow{a} q'} \quad \frac{p \xrightarrow{a} p' \quad X \stackrel{\text{def}}{=} p}{X \xrightarrow{a} p'} \\
\frac{}{\mathbf{1} \downarrow} \quad \frac{p \downarrow}{(p+q) \downarrow} \quad \frac{q \downarrow}{(p+q) \downarrow} \quad \frac{p \downarrow \quad X \stackrel{\text{def}}{=} p}{X \downarrow} \\
\frac{p \downarrow \quad q \downarrow}{(p; q) \downarrow} \quad \frac{p \xrightarrow{a} p'}{p; q \xrightarrow{a} p'; q} \quad \frac{p \downarrow \quad p \not\rightarrow \quad q \xrightarrow{a} q'}{p; q \xrightarrow{a} q'}
\end{array}$$

■ **Figure 1** Operational semantics for $\text{TSP}^i(\mathcal{A})$.

if there exist p_0, \dots, p_n such that $p = p_0$, $p_{i-1} \xrightarrow{a_i} p_i$ ($1 \leq i \leq n$) and $p_n = p'$. Also, we write $p \rightarrow p'$ for there exists $a \in \mathcal{A}$ such that $p \xrightarrow{a} p'$. Similarly, we write $p \twoheadrightarrow p'$ for there exists $w \in \mathcal{A}^*$ such that $p \xrightarrow{w} p'$ and say that p' is *reachable* from p .

It is well-known that transition system specifications with negative premises may not define a unique transition relation that agrees with provability from the transition system specification [15, 11, 14]. Indeed, in [7] it was already pointed out that the transition system specification in Fig. 1 gives rise to such anomalies, e.g., if Δ includes for X the defining equation $X \stackrel{\text{def}}{=} X; a.1 + \mathbf{1}$. For then, on the one hand, if $X \twoheadrightarrow$, according to the rules for sequencing and recursion we find that $X \xrightarrow{a} \mathbf{1}$, while on the other hand, the transition $X \xrightarrow{a} \mathbf{1}$ is not provable from the transition system specification.

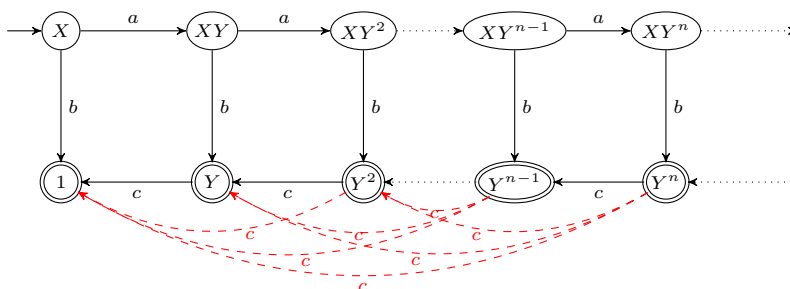
We remedy the situation by restricting our attention to *guarded* recursive specifications, i.e., we require that every occurrence of a process identifier in the definition of some (possibly different) process identifier occurs within the scope of an action prefix. If Δ is guarded, then it is straightforward to prove that the mapping S from process expressions to natural numbers inductively defined by $S(\mathbf{1}) = S(\mathbf{0}) = S(a.p) = 0$, $S(p_1 + p_2) = S(p_1; p_2) = S(p_1) + S(p_2) + 1$, and $S(X) = S(p)$ if $(X \stackrel{\text{def}}{=} p) \in \Delta$ gives rise to a so-called *stratification* S' from transitions to natural numbers defined by $S'(p \xrightarrow{a} p') = S(p)$ for all $a \in \mathcal{A}$ and process expressions p and p' . In [15] it is proved that whenever such a stratification exists, then the transition system specification defines a unique transition relation that agrees with provability in the transition system specification.

The operational rules in Fig. 1 deviate from the operational rules for the process theory $\text{TSP}(\mathcal{A})$ discussed in [2] in only two ways: to get the rules for $\text{TSP}^i(\mathcal{A})$, the symbol $;$ should be replaced by \cdot , and the negative premise $p \twoheadrightarrow$ should be removed from the right-most rule for sequencing. The replacement of $;$ by \cdot is, of course, insignificant; the removal of the negative premise $p \twoheadrightarrow$, however, does have a significant impact. The negative premise ensures that a sequencing can only proceed to execute its second argument when its first argument not only satisfies the acceptance predicate, but also cannot perform any further activity. The semantic difference between $;$ and \cdot is illustrated in the following example.

► **Example 1.** Consider the recursive specification

$$X \stackrel{\text{def}}{=} a.(XY) + b.\mathbf{1} \quad Y \stackrel{\text{def}}{=} c.\mathbf{1} + \mathbf{1} .$$

Depending on whether we interpret the concatenation of process identifiers as sequential composition (\cdot) or sequencing ($;$), we obtain the transition system shown in Fig. 2 with or without the dashed c -transitions. Note that, under the \cdot -interpretation, the phenomenon of *transparency* plays a role: from Y^n we have c -transitions to every Y^k with $k < n$, by



■ **Figure 2** The difference between $;$ and \cdot .

executing the c -transition of the k th occurrence of Y , thus skipping the first $k - 1$ occurrences of Y . This behaviour is prohibited by the negative premise in the rule for $;$, for, since $Y \xrightarrow{c} \mathbf{1}$, none of the occurrences of Y can be skipped.

► **Remark 2.** As Fig. 2 illustrates, the use of sequential composition (as opposed to sequencing) in guarded recursive specifications may give rise to an unbounded reachable branching degree (i.e., there need not be an upper bound on the branching degrees of states reachable from some particular state). As far as we know, this is the only process algebra without an operator for parallel composition that facilitates communication between parallel components that gives rise to unboundedly branching behaviour. It is this kind of behaviour that, e.g., cannot be exhibited by the transition system associated with a pushdown automaton [3].

We proceed to define when two process expressions are behaviourally equivalent.

► **Definition 3.** A binary relation R on the set of process expressions associated with $\text{TSP}^i(\mathcal{A}, \mathcal{P})$ is a bisimulation iff R is symmetric and for all p and q such that $(p, q) \in R$:

1. If $p \xrightarrow{a} p'$, then there exists a term q' , such that $q \xrightarrow{a} q'$, and $(p', q') \in R$.
2. If $p \downarrow$, then $q \downarrow$.

Process expressions p and q are bisimilar (notation: $p \Leftrightarrow q$) iff there exists a bisimulation R such that $(p, q) \in R$.

The operational rules presented in Fig 1 are in the so-called *panth format* from which it immediately follows that bisimilarity is a congruence [23].

► **Proposition 4.** The relation \Leftrightarrow is a congruence for $\text{TSP}^i(\mathcal{A}, \mathcal{P})$.

► **Remark 5.** Note that language equivalence is not a congruence for the sequencing operator: $a.b.\mathbf{1} + a.\mathbf{1}$ and $a.(b.\mathbf{1} + \mathbf{1})$ have the same language $\{ab, a\}$, but the language of $(a.b.\mathbf{1} + a.\mathbf{1}); c.\mathbf{1}$ is $\{abc, ac\}$ and the language of $a.(b.\mathbf{1} + \mathbf{1}); c.\mathbf{1}$ is $\{abc\}$.

3 Equational theory

In this section we shall consider $\text{TSP}^i(\mathcal{A}, \emptyset)$, i.e., the recursion-free fragment of the Theory of Sequential Processes. Let us abbreviate $\text{TSP}^i(\mathcal{A}, \emptyset)$ by $\text{TSP}^i(\mathcal{A})$.

For the purpose of concisely expressing equational properties, we shall use variables from some countably infinite set \mathcal{V} of variables. (These variables should be thought of as ranging over process expressions, and should not be confused with process identifiers.) The set of $\text{TSP}^i(\mathcal{A})$ -terms is generated by the following grammar ($a \in \mathcal{A}$, $x \in \mathcal{V}$):

$$t ::= \mathbf{0} \mid \mathbf{1} \mid a.t \mid t + t \mid t; t \mid x .$$

11:6 Sequencing and Intermediate Acceptance

A $\text{TSP}^i(\mathcal{A})$ -term is *closed* if it does not contain variables. Note that the set of closed $\text{TSP}^i(\mathcal{A})$ -terms coincides with the set of process expressions associated with $\text{TSP}^i(\mathcal{A}, \emptyset)$ in the previous section. A *closed substitution* is a mapping σ from variables to process expressions. If t is a $\text{TSP}^i(\mathcal{A})$ -term and σ is a closed substitution, then we denote by $\sigma(t)$ the process expression obtained by replacing every occurrence of a variable x in t by $\sigma(x)$.

Let t and u be $\text{TSP}^i(\mathcal{A})$ -terms; an expression of the form $t = u$ is called a $\text{TSP}^i(\mathcal{A})$ -*equation*. A $\text{TSP}^i(\mathcal{A})$ -equation $t = u$ is *valid* if $\sigma(t) \Downarrow \sigma(u)$ for every closed substitution σ . The *equational theory* of $\text{TSP}^i(\mathcal{A})$ is the set of all valid $\text{TSP}^i(\mathcal{A})$ -equations.

Let E be a set of valid equations and let $t = u$ be a $\text{TSP}^i(\mathcal{A})$ -equation. We shall write $E \vdash t = u$ if $t = u$ can be derived from the equations in E by means of the rules of equational logic. We wish to characterise the equational theory of $\text{TSP}^i(\mathcal{A})$ by giving a finite collection E of valid $\text{TSP}^i(\mathcal{A})$ -equations such that $E \vdash t = u$ for every valid $\text{TSP}^i(\mathcal{A})$ -equation $t = u$. Such a collection E is then referred to as a *finite basis* for the equational theory of $\text{TSP}^i(\mathcal{A})$; we say that an equational theory is *finitely based* if there exists a finite basis for it.

We shall prove two fundamental results pertaining to the equational theory of $\text{TSP}^i(\mathcal{A})$. First, we shall establish that there does not exist a finite basis for the equational theory of $\text{TSP}^i(\mathcal{A})$. Second, we shall prove that when an auxiliary operator is added, then the resulting *ground* equational theory (consisting only of all valid $\text{TSP}^i(\mathcal{A})$ -equations *without* variables) is finitely based. At the end of Section 3.2 we shall conclude with presenting some evidence for a conjecture that, even with the auxiliary operator added, the full equational theory (consisting of all valid $\text{TSP}^i(\mathcal{A})$ -equations *with* variables) is not finitely based.

3.1 $\text{TSP}^i(\mathcal{A})$ is not finitely based

A central axiom of the theory of $\text{TSP}(\mathcal{A})$ of [2] is the axiom $(x + y) \cdot z = x \cdot z + y \cdot z$, which expresses that sequential composition distributes from the right over alternative composition. For sequencing, the axiom is no longer valid in general as the following example illustrates.

► **Example 6.** Consider the process expressions

$$p \equiv (a.1 + 1) ; b.1 \text{ and } q \equiv a.1 ; b.1 + 1 ; b.;1 \text{ .}$$

(We write \equiv for syntactic equality of $\text{TSP}^i(\mathcal{A})$ -terms and reserve $=$ to express $\text{TSP}^i(\mathcal{A})$ -equations.) Note that, on the one hand, since $a.1 + 1 \xrightarrow{a} 1$, we have that $p \xrightarrow{b}$. On the other hand, since $1 \downarrow$ and $1 \rightarrow$, we do have that $1 ; b.1 \xrightarrow{b} 1$ and hence $q \xrightarrow{b} 1$. It follows that p and q are not bisimilar.

Note that, a fortiori, we have that $p \not\Downarrow a.1 ; b.1$. That the first argument of the sequencing operator satisfies the acceptance predicate has no effect, because the second argument of the sequencing operator does not satisfy the acceptance predicate. Thus, if the second argument of sequencing does not satisfy the acceptance predicate, then a 1 -summand in the first argument is redundant.

We shall prove that the redundancy of 1 at the left-hand side of sequencing cannot be finitely axiomatised without using an auxiliary operator. To this end, let us fix $\tilde{a}, \tilde{b} \in \mathcal{A}$ and consider the following infinite collection of valid equations ($n \in \mathbb{N}$):

$$(\tilde{a}.1 + 1) ; \sum_{i=1}^n \tilde{b}.(\tilde{b}.1 + 1)^i = \tilde{a}.1 ; \sum_{i=1}^n \tilde{b}.(\tilde{b}.1 + 1)^i \text{ .} \quad (e_n)$$

(For every natural number i process expression p , p^i denotes the *iterated* sequencing of p , inductively defined by $p^0 = 1$ and $p^{i+1} = p^i ; p$.)

Each of these equations expresses the redundancy of the occurrence of $\mathbf{1}$ in the subexpression $\tilde{a}.\mathbf{1} + \mathbf{1}$ on the left-hand side of the equation. For this redundancy it is important that the right-hand side of the sequencing operator, i.e., the process expression $\sum_{i=1}^n \tilde{b}.\tilde{b}.\mathbf{1} + \mathbf{1}$ ^{*i*}, does not satisfy the acceptance predicate, since it is a summation of \tilde{b} -prefixes without $\mathbf{1}$ -summand. That the number of summands is n will be used in our argument that (e_n) , for sufficiently large $n \in \mathbb{N}$, cannot be derived from a particular finite collection of valid equations. Instead of referring to the notion of number of summands, it is more convenient to refer to the notion of width that we shall now define.

► **Definition 7.** *The width of a process expression p , written as $\text{width}(p)$, is the cardinality of the set $\{p' \mid p \xrightarrow{a} p', \text{ for some } a \in A\}$. We extend the notion of width to $\text{TSP}^i(\mathcal{A})$ -terms by defining, for all $\text{TSP}^i(\mathcal{A})$ -terms t , that $\text{width}(t) = \text{width}(\sigma_{\mathbf{0}}(t))$ where $\sigma_{\mathbf{0}}$ denotes the closed substitution that maps all variables to $\mathbf{0}$.*

Note that variables do not contribute to the width of a $\text{TSP}^i(\mathcal{A})$ -term.

Suppose that E is a finite set of valid equations, and let $n \in \mathbb{N}$ exceed the maximum of the widths of all subterms occurring in the equations in E . To prove that (e_n) cannot be derived from E , we define a predicate Ψ_n on $\text{TSP}^i(\mathcal{A})$ -terms that is satisfied by the left-hand side (e_n) , but not by the right-hand side, and that is maintained by equational derivations from E .

► **Definition 8.** *Let p be a process expression. For every $n \in \mathbb{N}$, we define that $\Phi_n(p)$ holds iff $p \equiv p_1 ; p_2$ such that $p_1 \Leftrightarrow \tilde{a}.\mathbf{1} + \mathbf{1}$ and $p_2 \Leftrightarrow \sum_{i=1}^n \tilde{b}.\tilde{b}.\mathbf{1} + \mathbf{1}$ ^{*i*}. For every $n \in \mathbb{N}$, we define $\Psi_n(p)$ iff $p \Leftrightarrow (\tilde{a}.\mathbf{1} + \mathbf{1}) ; \sum_{i=1}^n \tilde{b}.\tilde{b}.\mathbf{1} + \mathbf{1}$ ^{*i*} and p has a summand $p_1 ; p_2$ such that one of the following cases holds:*

1. $\Phi_n(p_1 ; p_2)$.
2. $p_1 \Leftrightarrow \mathbf{1}$ and $\Psi_n(p_2)$.
3. $\Psi_n(p_1)$ and $p_2 \Leftrightarrow \mathbf{1}$.

The predicate Φ_n formalises a property satisfied by the left-hand side of (e_n) , but not by the right-hand side. The property Φ_n is, however, not preserved by equational derivations due to certain trivial syntactic manipulations involving, e.g., the idempotence of $+$, $\mathbf{0}$ being a neutral element for $+$ and $\mathbf{1}$ being a left- and right neutral element for sequencing (see Table 1 below). The definition of Ψ_n takes such syntactic manipulations into account. Note that the definition of Ψ_n is with recursion on the syntactic structure; it is well-defined since in the last two cases of its definition it is evaluated on a proper subterm.

In general, bisimilarity does not preserve width as defined above, but it does hold that if $p \Leftrightarrow q$ and there exist process expressions p_1, \dots, p_n such that $p \xrightarrow{a} p_i$ and $p_i \Leftrightarrow p_j$ implies $i = j$ for all $1 \leq i, j \leq n$, then $\text{width}(q) \geq n$. Note that the process expression $\sum_{i=1}^n \tilde{b}.\tilde{b}.\mathbf{1} + \mathbf{1}$ ^{*i*} has this property. We exploit it to argue that if t is a $\text{TSP}^i(\mathcal{A})$ -term such that $\text{width}(t) < n$ and σ is a closed substitution such that $\sigma(t) \Leftrightarrow \sum_{i=1}^n \tilde{b}.\tilde{b}.\mathbf{1} + \mathbf{1}$ ^{*i*}, then necessarily t has a variable summand, say x , such that $\sigma(x) \xrightarrow{\tilde{b}}$. This means that with a minor modification of σ , we obtain a substitution $\vartheta_{(\sigma,x)}$ such that $\vartheta_{(\sigma,x)}(t) \downarrow$. We define $\vartheta_{(\sigma,x)}$ as follows:

$$\vartheta_{(\sigma,x)}(y) = \begin{cases} \sigma(y) + \mathbf{1} & \text{if } y = x \\ \sigma(y) & \text{otherwise.} \end{cases}$$

The following lemma essentially applies this idea in a slightly more general situation, where $\sigma(t)$ satisfies Ψ_n .

11:8 Sequencing and Intermediate Acceptance

► **Lemma 9.** *Let t be a $\text{TSP}^i(\mathcal{A})$ -term and let n be a natural number such that $\text{width}(t') < n$ for every subterm t' of t . If $\Psi_n(\sigma(t))$ for some closed substitution σ , then there is a variable x such that $\vartheta_{(\sigma,x)}(t) \downarrow$ and either $\Psi_n(\sigma(x))$ or $\sigma(x) \xrightarrow{\tilde{b}}$.*

Proof. See the proof of Lemma 35 in Appendix A. ◀

The following lemma establishes the converse of Lemma 9.

► **Lemma 10.** *Let t be a $\text{TSP}^i(\mathcal{A})$ -term, let x be a variable, and let σ be a closed substitution. If $\sigma(t) \Leftrightarrow (\tilde{a}.\mathbf{1} + \mathbf{1}) ; \sum_{i=1}^n (\tilde{b}.\mathbf{1} ; (\tilde{b}.\mathbf{1} + \mathbf{1})^i)$, $\vartheta_{(\sigma,x)}(t) \downarrow$ and either $\Psi_n(\sigma(x))$ or $\sigma(x) \xrightarrow{\tilde{b}}$, then $\Psi_n(\sigma(t))$.*

Proof. See the proof of Lemma 40 in Appendix A. ◀

The following theorem states that if E is a set of valid equations and n exceeds the maximum of the widths of all subterms of the equations in E , then equational derivations from E preserve Ψ_n .

► **Theorem 11.** *Let E be a finite set of valid $\text{TSP}^i(\mathcal{A})$ -equations, and let n be a natural number such that for each axiom $t = u \in E$, for each subterm t' of t and each subterm u' of u , $\text{width}(t') < n$ and $\text{width}(u') < n$. Furthermore, let p and q be closed $\text{TSP}^i(\mathcal{A})$ -terms such that $E \vdash p = q$. It then holds that if $\Psi_n(p)$, then $\Psi_n(q)$.*

Proof. The proof is by induction on a derivation of the equation $p = q$ from E . So, we distinguish cases, according to the last rule used in this derivation, and assume that for each derivation of $p' = q'$ that is a sub-derivation of the derivation of $p = q$, if $\Psi_n(p')$ then $\Psi_n(q')$ (IH). Here we only consider the most interesting case in which the derivation consists of a substitution instance of an axiom in E .

If $p = q$ is a substitution instance of an axiom in E , then there exist $\text{TSP}^i(\mathcal{A})$ -terms t and u and a closed substitution σ such that $\sigma(t) = p$, $\sigma(u) = q$ and $t = u \in E$. If $\Psi_n(p)$, then $\Psi_n(\sigma(t))$ and thus $\sigma(t) \Leftrightarrow (\tilde{a}.\mathbf{1} + \mathbf{1}) ; \sum_{i=1}^n (\tilde{b}.\mathbf{1} ; (\tilde{b}.\mathbf{1} + \mathbf{1})^i)$. Since $t = u$ is sound with respect to bisimilarity, $\sigma(u) \Leftrightarrow \sigma(t) \Leftrightarrow (\tilde{a}.\mathbf{1} + \mathbf{1}) ; \sum_{i=1}^n (\tilde{b}.\mathbf{1} ; (\tilde{b}.\mathbf{1} + \mathbf{1})^i)$. Furthermore, by Lemma 9, there must be some variable x such that $\vartheta_{(\sigma,x)}(t) \downarrow$ and either $\Psi_n(\sigma(x))$ or $\sigma(x) \xrightarrow{\tilde{b}}$ for some closed $\text{TSP}^i(\mathcal{A})$ -term p . Hence, since $\vartheta_{(\sigma,x)}(t) \Leftrightarrow \vartheta_{(\sigma,x)}(u)$, also $\vartheta_{(\sigma,x)}(u) \downarrow$. Then, since $\sigma(u) \Leftrightarrow (\tilde{a}.\mathbf{1} + \mathbf{1}) ; \sum_{i=1}^n (\tilde{b}.\mathbf{1} ; (\tilde{b}.\mathbf{1} + \mathbf{1})^i)$, and $\vartheta_{(\sigma,x)}(u) \downarrow$, by Lemma 10, we conclude that $\Psi_n(\sigma(u))$ holds, and thus $\Psi_n(q)$ holds. ◀

We use Theorem 11 to prove that the equational theory of $\text{TSP}^i(\mathcal{A})$ is not finitely based by showing that no set E of valid $\text{TSP}^i(\mathcal{A})$ -equations can be a finite basis. To this end, let E be a finite set of valid $\text{TSP}^i(\mathcal{A})$ -equations. Then, since E has finitely many equations and the terms occurring on both sides of these equations each have finitely many subterms, there exists $n \in \mathbb{N}$ that exceeds the widths of all these subterms. By Theorem 11 we have that whenever $E \vdash p = q$ and $\Psi_n(p)$, then also $\Psi_n(q)$; it follows that $E \not\vdash (e_n)$. Since (e_n) is a valid $\text{TSP}^i(\mathcal{A})$ -equation, it follows that E is not a finite basis for the equational theory of $\text{TSP}^i(\mathcal{A})$. Thus, we obtain the following corollary.

► **Corollary 12.** *There does not exist a finite basis for $\text{TSP}^i(\mathcal{A})$.*

3.2 Ground-completeness with an auxiliary operator

By the *ground* equational theory of $\text{TSP}^i(\mathcal{A})$ we mean the set of all valid $\text{TSP}^i(\mathcal{A})$ -equations *without variables*. Note that, since the equations (e_n) do not include variables and the predicate Ψ_n is defined on process expressions, it immediately follows from Theorem 11 that the ground equational theory of $\text{TSP}^i(\mathcal{A})$ is not finitely based either. We proceed to extend $\text{TSP}^i(\mathcal{A})$ with a unary auxiliary operator NT and show that ground equational theory of the extension $\text{TSP}_{NT}^i(\mathcal{A})$ is finitely based.

The syntax of $\text{TSP}_{NT}^i(\mathcal{A})$ consists of the syntax of $\text{TSP}^i(\mathcal{A})$ with the unary operation NT added; this operation can be used both in the construction of process expressions associated with $\text{TSP}_{NT}^i(\mathcal{A})$ and of $\text{TSP}_{NT}^i(\mathcal{A})$ -terms. Intuitively, $NT(p)$ denotes the non-terminating part of p ; for example, $NT(a.p) = a.p$ and $NT(a.p + \mathbf{1}) = a.p$.

$$\frac{p \xrightarrow{a} p'}{NT(p) \xrightarrow{a} p'}$$

■ **Figure 3** The operational rule for NT .

The operational rule for NT is presented in Figure 3. The rule is in the panth format, so bisimilarity is a congruence also for the extended theory. Furthermore, from [22, Theorem 3.9] it follows that $\text{TSP}_{NT}^i(\mathcal{A})$ is an operational conservative extension of $\text{TSP}^i(\mathcal{A})$, meaning that $\text{TSP}^i(\mathcal{A})$ process expressions have the same operational semantics in the extended theory $\text{TSP}_{NT}^i(\mathcal{A})$.

■ **Table 1** A finite basis for the ground equational theory of $\text{TSP}_{NT}^i(\mathcal{A})$.

| | | | |
|---|-----|---|-----|
| $x + y$ | $=$ | $y + x$ | A1 |
| $x + (y + z)$ | $=$ | $(x + y) + z$ | A2 |
| $x + x$ | $=$ | x | A3 |
| $(x ; y) ; z$ | $=$ | $x ; (y ; z)$ | A5 |
| $x + \mathbf{0}$ | $=$ | x | A6 |
| $\mathbf{0} ; x$ | $=$ | $\mathbf{0}$ | A7 |
| $x ; \mathbf{1}$ | $=$ | x | A8 |
| $\mathbf{1} ; x$ | $=$ | x | A9 |
| $a.x ; y$ | $=$ | $a.(x ; y)$ | A10 |
| $NT(x + y) ; z$ | $=$ | $NT(x) ; z + NT(y) ; z$ | A11 |
| $(a.x + y + \mathbf{1}) ; NT(z)$ | $=$ | $(a.x + y) ; NT(z)$ | A12 |
| $(a.x + y + \mathbf{1}) ; (z + \mathbf{1})$ | $=$ | $(a.x + y) ; (z + \mathbf{1}) + \mathbf{1}$ | A13 |
| $NT(\mathbf{0})$ | $=$ | $\mathbf{0}$ | NT1 |
| $NT(\mathbf{1})$ | $=$ | $\mathbf{0}$ | NT2 |
| $NT(a.x)$ | $=$ | $a.x$ | NT3 |
| $NT(x + y)$ | $=$ | $NT(x) + NT(y)$ | NT4 |

Table 1 presents a finite collection of valid $\text{TSP}_{NT}^i(\mathcal{A})$ -equations. It includes the well-known axioms A1–3 and A5–10 adapted from $\text{TSP}(\mathcal{A})$ (see [2]). Note, however, that the axiom A4, which in $\text{TSP}(\mathcal{A})$ expresses distributivity from the right of sequencing over

11:10 Sequencing and Intermediate Acceptance

alternative composition, has been omitted since it is not valid. It has been replaced by axiom A11, which, intuitively, expresses that sequencing distributes from the right over alternative composition only if the alternative composition does not satisfy the acceptance predicate. Axioms A12 and A13 allows us to eliminate redundant occurrences of $\mathbf{1}$ at the left-hand side of sequencing. Finally, axioms NT1–4 express the interaction of NT with the constants $\mathbf{0}$ and $\mathbf{1}$, action prefix and alternative composition.

For detailed proofs that the axioms in Table 1 are valid we refer to [8]. We shall, henceforth, write $\text{TSP}_{NT}^i(\mathcal{A}) \vdash t = u$ if the $\text{TSP}_{NT}^i(\mathcal{A})$ -equation $t = u$ can be derived from the axioms in Table 1 using the rules of equational logic. To prove that the axioms in Table 1 constitute a finite basis for the ground equational theory of $\text{TSP}_{NT}^i(\mathcal{A})$, we use the following elimination theorem.

► **Theorem 13.** *For every process expression p associated with $\text{TSP}_{NT}^i(\mathcal{A})$ there exists a process expression q without occurrences of $;$ and NT such that $\text{TSP}_{NT}^i(\mathcal{A}) \vdash p = q$.*

In [2, Theorem 4.4.12] it is proved that axioms A1–3 and A6 constitute a finite basis for the ground equational theory of $\text{BSP}(\mathcal{A})$, which is obtained from $\text{TSP}_{NT}^i(\mathcal{A})$ by removing $;$ and NT . Hence, we get the following corollary from Theorem 13.

► **Corollary 14.** *The axioms in Table 1 constitute a finite basis for the ground equational theory of $\text{TSP}_{NT}^i(\mathcal{A})$.*

The axioms in Table 1 do not constitute a finite basis for the full equational theory of $\text{TSP}_{NT}^i(\mathcal{A})$. For example, it is easy to see that the valid equation $NT(NT(x)) = NT(x)$ cannot be derived from $\text{TSP}_{NT}^i(\mathcal{A})$. We proceed to argue that, although the ground equational theory of $\text{TSP}_{NT}^i(\mathcal{A})$ is finitely based, the full equational theory of $\text{TSP}_{NT}^i(\mathcal{A})$ is not; the argument will be very similar to the argument showing that $\text{TSP}_{NT}^i(\mathcal{A})$ is not finitely based.

Consider the equation:

$$(x + \mathbf{1}) ; x = x ; x . \quad (1)$$

To see that it is valid, note that the symmetric closure of the relation

$$R = \{((p + \mathbf{1}) ; p, p ; p), (p, p) \mid p \text{ a } \text{TSP}_{NT}^i(\mathcal{A}) \text{ process expression}\}$$

is a bisimulation relation.

Recall the equations (e_n) used in Section 3.1 to show that the equational theory of $\text{TSP}^i(\mathcal{A})$ is not finitely based. For the redundancy of the $\mathbf{1}$ -summand in the left-hand side of the sequencing operator it is essential that the left-hand side also admits a transition while the right-hand side does not satisfy the acceptance predicate. Equation (1) above does not satisfy this property for every closed substitution. Nevertheless, the $\mathbf{1}$ -summand is redundant, due to the fact that x appears in both arguments of the sequencing operator. The idea can be generalised, resulting in the infinite collection of valid equations $n \in \mathbb{N}$:

$$(x + \mathbf{1}) ; \sum_{i=1}^n (x ; (a.\mathbf{1} + \mathbf{1})^i) = x ; \sum_{i=1}^n (x ; (a.\mathbf{1} + \mathbf{1})^i) . \quad (e'_n)$$

Similarly to the equations (e_n) used in Section 3.1, the size of the right hand side of this equation is not bounded. We conjecture that by similar reasoning as used in Section 3.1 it can be argued that there does not exist a finite set of valid $\text{TSP}_{NT}^i(\mathcal{A})$ -equations from which the equations e'_n can be derived for all $n \in \mathbb{N}$.

4 Decidability

Christensen, Hüttel and Stirling have established that bisimilarity is decidable for processes definable by means of a guarded recursive BPA specification [13], where BPA can be thought of as $\text{TSP}^i(\mathcal{A})$ without intermediate acceptance. Our goal in this section is to extend that decidability result to $\text{TSP}^i(\mathcal{A})$. Our proof closely follows the presentation of the decidability proof for BPA in [12], and we shall focus on the extension and skip over parts that are similar.

The starting point is the presupposed $\text{TSP}^i(\mathcal{A}, \mathcal{P})$ recursive specification Δ , which is finite since we have assumed that \mathcal{P} is finite. The decision problem we wish to solve is: Given any two process expressions p and q does it hold that $p \Leftrightarrow q$? We shall first recall a few standard observations to simplify the formulation of the decision problem.

The first observation is that we may assume, without loss of generality, that p and q are both process identifiers. For if not then we could first solve the decision problem for process identifiers, and then decide $p \Leftrightarrow q$ by considering $\text{TSP}^i(\mathcal{A}, \mathcal{P}')$, where \mathcal{P}' is \mathcal{P} with two new process identifiers X and Y added and Δ' is Δ with the two extra defining equations $X \stackrel{\text{def}}{=} p$ and $Y \stackrel{\text{def}}{=} q$, and determine whether $X \Leftrightarrow Y$.

The second observation is that we may assume, again without loss of generality, that Δ is in so-called *Greibach Normal Form* (GNF): for every defining equation $(X \stackrel{\text{def}}{=} p) \in \Delta$ we have that

$$p \equiv \sum_{i=1}^n a_i \cdot \alpha_i (+\mathbf{1}) . \quad (2)$$

Here we assume that $n \in \mathbb{N}$ (recall our convention that the empty summation denotes $\mathbf{0}$), $\alpha_i \in \mathcal{P}^*$, and $(+\mathbf{1})$ denotes an optional $\mathbf{1}$ -summand. We refer to [8] for the description of an effective procedure that associates with every recursive specification Δ over $\text{TSP}^i(\mathcal{A}, \mathcal{P})$ a set $\mathcal{P}' \supseteq \mathcal{P}$ and a recursive specification Δ' over $\text{TSP}^i(\mathcal{A}, \mathcal{P}')$ in GNF such that for all $X, Y \in \mathcal{P}$ we have that $X \Leftrightarrow Y$ with respect to Δ if, and only if, $X \Leftrightarrow Y$ with respect to Δ' . The advantage of assuming that Δ is in GNF is that then every process expression reachable (by following the transition relation) from a process identifier associated with $\text{TSP}^i(\mathcal{A}, \mathcal{P})$ is an element of \mathcal{P}^* .

The third observation is that it is semi-decidable whether $p \not\equiv q$. This is a straightforward consequence of the well-known fact that for image-finite transition systems there is a stratified characterisation of bisimilarity; see [8] for such a characterisation taking the acceptance predicate into account. Therefore, to solve the aforementioned decision problem, it suffices to argue that bisimilarity is semi-decidable.

The argument presented in [12] to show that bisimilarity is semi-decidable for BPA then proceeds by showing that process identifiers X and Y are bisimilar if, and only if, there exists a *finite bisimulation base*¹ that contains the pair (X, Y) , and that it is semi-decidable whether a finite binary relation on \mathcal{P}^* is a bisimulation base. It then follows that $X \Leftrightarrow Y$ is semi-decidable: enumerate all finite binary relations on \mathcal{P}^* containing the pair (X, Y) and check, in parallel, whether one of them is a bisimulation base.

We adapt the definition of bisimulation base, originally from [13], to our setting with an acceptance predicate. It uses the following auxiliary notation: if R is a binary relation on \mathcal{P}^* ,

¹ Note that a *bisimulation base* is a *bisimulation up to congruence* with respect to the operation of concatenation on finite sequences of process identifiers [20].

11:12 Sequencing and Intermediate Acceptance

then we denote by $\stackrel{R}{\equiv}$ the least equivalence relation that contains R and all pairs $(\alpha\alpha', \beta\beta')$ whenever it contains the pairs (α, β) and (α', β') .

► **Definition 15.** A binary relation R on \mathcal{P}^* is a bisimulation base if, and only if, R is symmetric and for all pairs $(\alpha, \beta) \in R$ and all $a \in \mathcal{A}$, it holds that:

- if $\alpha \xrightarrow{a} \alpha'$, then $\beta \xrightarrow{a} \beta'$ for some β' such that $\alpha' \stackrel{R}{\equiv} \beta'$; and
- if $\alpha \downarrow$, then $\beta \downarrow$.

Our goal will be to show that there exists a finite bisimulation base R such that $\alpha \stackrel{R}{\equiv} \beta$ if, and only if, $\alpha \Leftrightarrow \beta$ for all $\alpha, \beta \in \mathcal{P}^*$. The argument relies on a partitioning of the set of process identifiers into *normed* and *unnormed* process identifiers.

► **Definition 16.** Let p be a $\text{TSP}^i(\mathcal{A}, \mathcal{P})$ process expression. The norm $n(p)$ of p is the length of a shortest transition sequence from p to a process expression bisimilar to $\mathbf{1}$ if such a sequence exists, and ∞ otherwise, i.e.,

$$n(p) = \min \left(\{ |w| \mid \exists p'. p \xrightarrow{w} p' \wedge p' \Leftrightarrow \mathbf{1} \} \cup \{ \infty \} \right) .$$

A process expression p is *normed* if $n(p) < \infty$; otherwise it is *unnormed*. We denote by \mathcal{P}_n the set of all normed process identifiers and by \mathcal{P}_u the set of all unnormed process identifiers.

In the case of BPA, which does not have intermediate acceptance, the following three properties hold for all sequences of process identifiers α, β and γ :

1. if α is unnormed, then $\alpha\beta \Leftrightarrow \alpha$;
2. $|\alpha| \leq n(\alpha)$; and
3. if α, β and γ are normed, then $\alpha\gamma \Leftrightarrow \beta\gamma$ implies $\alpha \Leftrightarrow \beta$.

These properties are crucial for pruning the cardinality of the bisimulation base. The following example illustrate that neither of these properties holds in our setting with intermediate acceptance:

► **Example 17.** Consider the following recursive specification in GNF:

$$\begin{array}{lll} X \stackrel{\text{def}}{=} a.YWZ + a.YW + a.ZZ + a.UV & Z \stackrel{\text{def}}{=} b.\mathbf{1} & V \stackrel{\text{def}}{=} \mathbf{0} \\ Y \stackrel{\text{def}}{=} b.\mathbf{1} + \mathbf{1} & U \stackrel{\text{def}}{=} b.U + \mathbf{1} & W \stackrel{\text{def}}{=} \mathbf{0} + \mathbf{1} \end{array}$$

Then we have that U is unnormed, but $UV \not\stackrel{R}{\equiv} U$ since $U \downarrow$ whereas $UV \not\downarrow$, refuting the first property. Furthermore, $|YWZ| = 3 > 2 = n(YWZ)$, refuting the second property. And finally $YWZ \Leftrightarrow ZZ$, but $YW \not\stackrel{R}{\equiv} Z$, refuting the third property.

Note that the sequences used in Example 17 to refute the properties above all suffer from some form of redundant intermediate acceptance.

Violation of the first property can only be due to the presence of a process identifier that is bisimilar to $\mathbf{1}$. Note that, in a recursive specification in GNF, a process identifier X is bisimilar to $\mathbf{1}$ if, and only if, $(X \stackrel{\text{def}}{=} \mathbf{0} + \mathbf{1}) \in \Delta$; let us call such a process identifier a *1-identifier*. Whether some process identifier is a 1-identifier can easily be decided. Furthermore, occurrences of 1-identifiers can simply be eliminated from the right-hand sides of defining equations of other process identifiers. Thus, it remains to solve the decision problem for recursive specifications in GNF without 1-identifiers.

Our main contribution in the remainder of this section will be the notion of *Acceptance Irredundant Greibach Normal Form* (AIGNF), a special variant of GNF that precludes redundant intermediate acceptance from sequences reachable from process identifiers. We

shall prove that it is enough to solve the decision problem for recursive specifications in AIGNF and then show that the argument for the existence of a finite bisimulation base of [13] works for such recursive specifications.

4.1 Acceptance Irredundant Greibach Normal Form

We partition the set of process identifiers \mathcal{P} into sets $\mathcal{P}_\downarrow = \{X \in \mathcal{P} \mid X \downarrow\}$ and $\mathcal{P}_\not\downarrow = \{X \in \mathcal{P} \mid X \not\downarrow\}$. Furthermore, we define the set $\overline{\mathcal{P}}_\not\downarrow$ of *hereditarily non-terminating* process identifiers as the largest subset of $\mathcal{P}_\not\downarrow$ such that for all $X \in \overline{\mathcal{P}}_\not\downarrow$ we have that if $(X \stackrel{\text{def}}{=} p) \in \Delta$ and Y is a process identifier occurring in p , then $Y \in \overline{\mathcal{P}}_\not\downarrow$. The set $\overline{\mathcal{P}}_\not\downarrow$ can be computed iteratively: start with $\mathcal{P}' = \mathcal{P}_\not\downarrow$ and in every iteration remove from \mathcal{P}' all process identifiers X such that $(X \stackrel{\text{def}}{=} p) \in \Delta$ and p has an occurrence of some process identifier Y with $Y \notin \mathcal{P}'$ until a fixed point is reached (i.e., nothing can be removed from \mathcal{P}' anymore). We shall say that $\alpha \in \mathcal{P}^*$ is *acceptance irredundant* if $\alpha \in \overline{\mathcal{P}}_\not\downarrow^* \mathcal{P}_\downarrow^* \cup \mathcal{P}_\downarrow^*$.

► **Definition 18.** A recursive specification Δ is in Acceptance Irredundant Greibach Normal Form (AIGNF) if for every defining equation $(X \stackrel{\text{def}}{=} p) \in \Delta$ we have that $p \equiv \mathbf{0}$ or

$$p \equiv \sum_{i=1}^n a_i \cdot \alpha_i (+\mathbf{1}) ,$$

with $n \in \mathbb{N}^+$ and each α_i acceptance irredundant.

The following example illustrates how a recursive specification in GNF and without 1-identifiers can be transformed into AIGNF.

► **Example 19.** Consider the following recursive specification in GNF:

$$\begin{aligned} X &\stackrel{\text{def}}{=} a.YZ + a.Y + a.ZZ + a.UV & Z &\stackrel{\text{def}}{=} b.\mathbf{1} \\ Y &\stackrel{\text{def}}{=} b.\mathbf{1} + \mathbf{1} \end{aligned}$$

As $Z \not\downarrow$, the intermediate acceptance of Y in $a.YZ$ is redundant. We cannot simply remove it from the definition of Y , however, since in $a.Y$ the intermediate acceptance of Y is not redundant. Instead, we introduce a fresh variable \bar{Y} , that is defined as Y but without the intermediate acceptance. Then, we replace all occurrences of Y of which the intermediate acceptance is redundant with \bar{Y} , resulting in:

$$\begin{aligned} X &\stackrel{\text{def}}{=} a.\bar{Y}Z + a.Y + a.ZZ + a.UV & Z &\stackrel{\text{def}}{=} b.\mathbf{1} \\ Y &\stackrel{\text{def}}{=} b.\mathbf{1} + \mathbf{1} & \bar{Y} &\stackrel{\text{def}}{=} b.\mathbf{1} \end{aligned}$$

The idea explained in the preceding example can be exploited to prove the following proposition.

► **Proposition 20.** For every recursive specification Δ over $\text{TSP}^i(\mathcal{A}, \mathcal{P})$ in GNF without 1-identifiers there exist $\mathcal{P}' \supseteq \mathcal{P}$ and a recursive specification Δ' in AIGNF over $\text{TSP}^i(\mathcal{A}, \mathcal{P}')$ such that for all $X, Y \in \mathcal{P}$ we have that $X \Leftrightarrow Y$ with respect to Δ if, and only if, $X \Leftrightarrow Y$ with respect to Δ' .

Let $\alpha \in \mathcal{P}^*$; we say that α is Δ -reachable if there exists $X \in \mathcal{P}$ such that $X \twoheadrightarrow \alpha$. If Δ is in AIGNF, then it can be shown that all Δ -reachable sequences are acceptance irredundant. Hence, for recursive specifications in AIGNF we now get the three properties needed for the proof that there exists a finite bisimulation base.

- **Proposition 21.** *If Δ is in AIGNF, then for all acceptance irredundant sequences α, β, γ :*
1. *if α is unnormed, then $\alpha\beta \Leftrightarrow \alpha$;*
 2. *$|\alpha| \leq n(\alpha)$; and*
 3. *if α, β and γ are normed, then $\alpha\gamma \Leftrightarrow \beta\gamma$ implies $\alpha \Leftrightarrow \beta$.*

Proof. See Appendix B. ◀

4.2 The existence of a finite bisimulation base

By the first item of Proposition 21, we can, without loss of generality, assume that all sequences of variables appearing in the right-hand sides of the defining equations in our presupposed recursive specification Δ in AIGNF are elements of $\mathcal{P}_n^* \cup \mathcal{P}_n^* \mathcal{P}_u$. Then all Δ -reachable sequences will not only be acceptance irredundant, but also elements of $\mathcal{P}_n^* \cup \mathcal{P}_n^* \mathcal{P}_u$.

The definition of the finite bisimulation base relies on decomposing sequences.

► **Definition 22.** *A pair $(X\alpha, Y\beta)$ satisfying $X\alpha \Leftrightarrow Y\beta$ is decomposable if X and Y are normed, and there exists γ such that*

- *$X \twoheadrightarrow \gamma, X \Leftrightarrow Y\gamma$ and $\gamma\alpha \Leftrightarrow \beta$; or*
- *$Y \twoheadrightarrow \gamma, Y \Leftrightarrow X\gamma$ and $\gamma\beta \Leftrightarrow \alpha$.*

Two pairs $(X\alpha, Y\beta)$ and $(X\alpha', Y\beta')$ are *distinct* if $\alpha \not\Leftrightarrow \alpha'$ or $\beta \not\Leftrightarrow \beta'$. A crucial step towards a finite bisimulation base consists of establishing that a relation containing all indecomposable pairs $(X\alpha, Y\beta)$, where $X\alpha, Y\beta \in \mathcal{P}_n^* \cup \mathcal{P}_n^* \mathcal{P}_u$ are acceptance irredundant sequences such that $X\alpha \Leftrightarrow Y\beta$ is necessarily finite.

For the definition of a finite bisimulation base we now need just one more definition, which allows us to choose appropriate candidates among non-distinct indecomposable pairs.

► **Definition 23.** *The finite prefix norm $n_f(\alpha)$ of α is defined as follows:*

$$n_f(\alpha) = \max(\{n(\beta) \mid n(\beta) < \infty \text{ and } \alpha = \beta\gamma \text{ for some } \gamma\}).$$

The pre-order \preceq on pairs is defined as:

$$(\alpha_1, \alpha_2) \preceq (\beta_1, \beta_2) \text{ iff } \max(n_f(\alpha_1), n_f(\alpha_2)) \leq \max(n_f(\beta_1), n_f(\beta_2)).$$

In the following two lemmas, adapted from [12, Lemmas 28 and 29], a relaxed form of cancellation is established for Δ -reachable sequences of process identifiers.

► **Lemma 24.** *If $\alpha \Leftrightarrow \gamma\alpha$ and $\beta \Leftrightarrow \gamma\beta$ for some $\gamma \not\equiv \mathbf{1}$ and acceptance irredundant $\gamma\alpha$ and $\gamma\beta$, then $\alpha \Leftrightarrow \beta$.*

Using this result, we will show a form of cancellation for (potentially unnormed) acceptance irredundant sequences, if $\alpha\gamma \Leftrightarrow \beta\gamma$ for infinitely many non-bisimilar γ .

► **Lemma 25.** *Let $\alpha, \beta \in \mathcal{P}^*$. If for infinitely many non-bisimilar $\gamma \in \mathcal{P}^*$ such that $\alpha\gamma$ and $\beta\gamma$ are acceptance irredundant it holds that $\alpha\gamma \Leftrightarrow \beta\gamma$, then $\alpha \Leftrightarrow \beta$.*

The following lemma is an adaptation of [12, Lemma 32] to our setting.

► **Lemma 26.** *For all $X, Y \in \mathcal{P}$, every set R of the form*

$$\{(X\alpha, Y\beta) \mid X\alpha, Y\beta \in \mathcal{P}_n^* \cup \mathcal{P}_n^* \mathcal{P}_u \text{ acceptance irredundant sequences,} \\ X\alpha \Leftrightarrow Y\beta, \text{ and } (X\alpha, Y\beta) \text{ indecomposable}\}$$

and contains only distinct pairs must be finite.

We now have everything in place to prove the main result of this section.

► **Theorem 27.** *Let $R_1 = \{(X, \alpha) \mid X \in \mathcal{P}_n, \alpha \in \mathcal{P}_n \text{ such that } X \Leftrightarrow \alpha\}$, and let R_2 be the largest relation of the form*

$$\{(X\alpha, Y\beta) \mid X\alpha, Y\beta \in \mathcal{P}_n^* \cup \mathcal{P}_n^* \mathcal{P}_u \text{ are acceptance irredundant,} \\ X\alpha \Leftrightarrow Y\beta, \text{ and } (X\alpha, Y\beta) \text{ indecomposable}\}$$

containing only distinct pairs and minimal elements with respect to \preceq . Then the symmetric closure R of $R_1 \cup R_2$ is finite and satisfies $\alpha \stackrel{R}{\equiv} \beta$ if and only if $\alpha \Leftrightarrow \beta$ for all acceptance irredundant sequences $\alpha, \beta \in \mathcal{P}_n^ \cup \mathcal{P}_n^* \mathcal{P}_u$.*

Proof. Since Δ is in AIGNF, we have $|\alpha| \leq n(\alpha)$. Hence, since X is normed and $n(X) = n(\alpha)$ we have $|\alpha| \leq n(X)$, and thus α has a finite maximum length. Hence, there can only be finitely many such α as \mathcal{P} is finite. It follows that R_1 is finite. Furthermore, by Lemma 26, R_2 is finite. So R is finite. Hence, since \Leftrightarrow is a congruence for $\text{TSP}^i(\mathcal{A})$, we have $\stackrel{R}{\equiv} \subseteq \Leftrightarrow$. It remains to show is that $\stackrel{R}{\equiv} \supseteq \Leftrightarrow$. We prove by induction on \preceq that $X\alpha \Leftrightarrow Y\beta$ implies $X\alpha \stackrel{R}{\equiv} Y\beta$, for all acceptance irredundant sequences $X\alpha, Y\beta \in \mathcal{P}_n^* \cup \mathcal{P}_n^* \mathcal{P}_u$.

Suppose that $(X\alpha, Y\beta)$ is decomposable, then $X, Y \in \mathcal{P}_n$ and, without loss of generality, assume that $X \twoheadrightarrow \gamma$ such that $X \Leftrightarrow Y\gamma$ and $\gamma\alpha \Leftrightarrow \beta$. Then, $n_f(\gamma\alpha) < n_f(Y\gamma\alpha) = n_f(X\alpha)$ and $n_f(\beta) < n_f(Y\beta)$, so $(\gamma\alpha, \beta) \prec (X\alpha, Y\beta)$. Furthermore, since $X \twoheadrightarrow \gamma$, $X\alpha \twoheadrightarrow \gamma\alpha$ and thus $\gamma\alpha \in \mathcal{P}_n^* \cup \mathcal{P}_n^* \mathcal{P}_u$ and $\gamma\alpha$ is acceptance irredundant. Moreover, since $Y\beta \in \mathcal{P}_n^* \cup \mathcal{P}_n^* \mathcal{P}_u$, $Y\beta$ is acceptance irredundant and $Y \in \mathcal{P}_n$, it follows that $\beta \in \mathcal{P}_n^* \cup \mathcal{P}_n^* \mathcal{P}_u$ and β is acceptance irredundant, and hence by induction $\gamma\alpha \stackrel{R}{\equiv} \beta$. Finally, since $\gamma\alpha$ is acceptance irredundant, γ is acceptance irredundant, and therefore $Y\gamma$ is acceptance irredundant. Hence, $(X, Y\gamma) \in R_1$ and thus $X\alpha \stackrel{R}{\equiv} Y\gamma\alpha \stackrel{R}{\equiv} Y\beta$.

Now, suppose that $(X\alpha, Y\beta)$ is not decomposable. Then $(X\alpha', Y\beta') \in R_2$ for some $\alpha' \Leftrightarrow \alpha$ and $\beta' \Leftrightarrow \beta$ with $(\alpha', \beta') \preceq (\alpha, \beta)$. We distinguish three cases.

- If $X, Y \in V_n$, then $(\alpha, \beta), (\alpha', \beta') \prec (X\alpha, Y\beta)$, so $(\alpha, \alpha'), (\beta, \beta') \prec (X\alpha, Y\beta)$. Hence, by induction $\alpha \stackrel{R}{\equiv} \alpha'$ and $\beta \stackrel{R}{\equiv} \beta'$, so $X\alpha \stackrel{R}{\equiv} X\alpha' RY\beta' \stackrel{R}{\equiv} Y\beta$.
- If $X \in V_n$ and $Y \in V_u$, then since $\beta \equiv X_1 \dots X_n$ for some $n \geq 0$ and $YX_i \stackrel{R}{\equiv} Y$ for each $0 \leq i \leq n$, we find $Y\beta \stackrel{R}{\equiv} Y$. Furthermore, $n_f(\alpha') \leq n_f(\alpha) < n_f(X\alpha)$, so $(\alpha, \alpha') \prec (X\alpha, Y)$. Hence, by induction $\alpha \stackrel{R}{\equiv} \alpha'$, and since $(X\alpha', Y) \in R_2$ we find $X\alpha \stackrel{R}{\equiv} X\alpha' \stackrel{R}{\equiv} Y \stackrel{R}{\equiv} Y\beta$. A symmetric argument applies for the case when $X \in V_u$ and $Y \in V_n$.
- If $X, Y \in V_u$, then since $\alpha \equiv X_1 \dots X_n$ for some $n \geq 0$ and $XX_i \stackrel{R}{\equiv} X$ for each $0 \leq i \leq n$, we find $X\alpha \stackrel{R}{\equiv} X$. Similarly, we find $Y\beta \stackrel{R}{\equiv} Y$ and thus since $(X, Y) \in R_2$, we derive $X\alpha \stackrel{R}{\equiv} X \stackrel{R}{\equiv} Y \stackrel{R}{\equiv} Y\beta$. ◀

It follows from Theorem 27 that bisimilarity is semi-decidable, and since also non-bisimilarity is semi-decidable, we obtain the following corollary.

► **Corollary 28.** *Bisimilarity is decidable for all processes definable by means of a finite guarded recursive specification over $\text{TSP}^i(\mathcal{A}, \mathcal{P})$.*

5 Conclusion

We have considered a variant of the Theory of Sequential Processes proposed in [7] in which sequential composition is replaced by sequencing. The distinguishing feature of the resulting process theory is that it includes the notion of intermediate acceptance relevant for the

theory of automata and formal languages, without also including the complications that arise from transparency. (We should mention here that the variant of successful termination considered by Aceto and Hennessy in [1] also does not lead to transparency, but in their theory a non-deterministic choice successfully terminates only if *both* arguments successfully terminate, and hence it does not have intermediate acceptance.)

We have presented a finite axiomatisation of the ground equational theory of the recursion-free fragment of the Theory of Sequential Processes using the auxiliary operator NT and proved that a finite axiomatisation without auxiliary operators does not exist.

Processes definable by means of a finite guarded recursive specification over $TSP^i(\mathcal{A})$ may rightfully be referred to as *context-free processes*. Indeed, the language of a process definable by means of a finite guarded recursive specification is context-free, and for every context-free language there is a process definable by a finite guarded recursive specification over $TSP^i(\mathcal{A})$ with that language. In [7] it was already proved that every context-free process is bisimilar to a pushdown process. Here we have proved that bisimilarity is decidable for all context-free processes, extending the seminal result of Christensen, Hüttel and Stirling [13] with intermediate acceptance.

It follows from the work of Moller [19] that not every pushdown process is context-free. We conjecture that extending $TSP^i(\mathcal{A})$ with propositional signals suffices to facilitate the definability of all pushdown processes. This will be the topic of a forthcoming paper.

Another interesting remaining open problem is whether bisimilarity is also decidable for the variant of the Theory of Sequential Processes discussed in [2]. In [8] it is argued that properties 2 and 3 of Proposition 21 do not hold in this case, and it seems considerably more difficult to deal with the ensuing complications.

References

- 1 Luca Aceto and Matthew Hennessy. Termination, Deadlock, and Divergence. *J. ACM*, 39(1):147–187, 1992. doi:10.1145/147508.147527.
- 2 Jos C. M. Baeten, Twan Basten, and Michel Reniers. *Process algebra: equational theories of communicating processes*, volume 50. Cambridge University Press, 2010.
- 3 Jos C. M. Baeten, Pieter J. L. Cuijpers, Bas Luttik, and P. J. A. van Tilburg. A Process-Theoretic Look at Automata. In Farhad Arbab and Marjan Sirjani, editors, *Proceedings of FSEN 2009*, volume 5961 of *LNCS*, pages 1–33. Springer, 2009. doi:10.1007/978-3-642-11623-0_1.
- 4 Jos C. M. Baeten, Pieter J. L. Cuijpers, and P. J. A. van Tilburg. A Context-Free Process as a Pushdown Automaton. In Franck van Breugel and Marsha Chechik, editors, *Proceedings of CONCUR 2008*, volume 5201 of *LNCS*, pages 98–113. Springer, 2008. doi:10.1007/978-3-540-85361-9_11.
- 5 Jos C. M. Baeten, Bas Luttik, Tim Muller, and Paul van Tilburg. Expressiveness modulo bisimilarity of regular expressions with parallel composition. *Mathematical Structures in Computer Science*, 26(6):933–968, 2016. doi:10.1017/S0960129514000309.
- 6 Jos C. M. Baeten, Bas Luttik, and Paul van Tilburg. Reactive Turing machines. *Inf. Comput.*, 231:143–166, 2013. doi:10.1016/j.ic.2013.08.010.
- 7 Jos C. M. Baeten, Bas Luttik, and Fei Yang. Sequential Composition in the Presence of Intermediate Termination (Extended Abstract). In Kirstin Peters and Simone Tini, editors, *Proceedings of EXPRESS/SOS 2017*, volume 255 of *EPTCS*, pages 1–17, 2017. doi:10.4204/EPTCS.255.1.
- 8 Astrid Belder. Decidability of bisimilarity and axiomatisation for sequential processes in the presence of intermediate termination. Master’s thesis, Eindhoven University of Technology, 2018. Available from <https://research.tue.nl/en/studentTheses/decidability-of-bisimilarity-and-axiomatisation-for-sequential-pr>.

- 9 Jan A. Bergstra and Jan Willem Klop. Process Algebra for Synchronous Communication. *Information and Control*, 60(1-3):109–137, 1984. doi:10.1016/S0019-9958(84)80025-X.
- 10 Bard Bloom. When is Partial Trace Equivalence Adequate? *Formal Asp. Comput.*, 6(3):317–338, 1994. doi:10.1007/BF01215409.
- 11 Roland N. Bol and Jan Friso Groote. The Meaning of Negative Premises in Transition System Specifications. *J. ACM*, 43(5):863–914, 1996. doi:10.1145/234752.234756.
- 12 Olaf Burkart, Didier Caucal, Faron Moller, and Bernhard Steffen. Verification on Infinite Structures. In J.A. Bergstra, A.J. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, pages 545–623. Elsevier, 2001.
- 13 Søren Christensen, Hans Hüttel, and Colin Stirling. Bisimulation Equivalence is Decidable for All Context-Free Processes. *Inf. Comput.*, 121(2):143–148, 1995. doi:10.1006/inco.1995.1129.
- 14 Rob J. van Glabbeek. The meaning of negative premises in transition system specifications II. *J. Log. Algebr. Program.*, 60-61:229–258, 2004. doi:10.1016/j.jlap.2004.03.007.
- 15 Jan Friso Groote. Transition System Specifications with Negative Premises. *Theor. Comput. Sci.*, 118(2):263–299, 1993. doi:10.1016/0304-3975(93)90111-6.
- 16 Stephen C. Kleene. Representation of Events in Nerve Nets and Finite Automata. *Automata Studies*, pages 3–41, 1956.
- 17 Hans Leiß. Towards Kleene Algebra with Recursion. In Egon Börger, Gerhard Jäger, Hans Kleine Büning, and Michael M. Richter, editors, *Proceedings of CSL '91*, volume 626 of *LNCS*, pages 242–256. Springer, 1991. doi:10.1007/BFb0023771.
- 18 R. Milner. *Communication and Concurrency*. Prentice Hall, Englewood Cliffs, 1989.
- 19 Faron Moller. Infinite Results. In Ugo Montanari and Vladimiro Sassone, editors, *Proceedings of CONCUR '96*, volume 1119 of *Lecture Notes in Computer Science*, pages 195–216. Springer, 1996. doi:10.1007/3-540-61604-7_56.
- 20 Damien Pous and Davide Sangiorgi. Enhancements of the bisimulation proof method. In Davide Sangiorgi and Jan Rutten, editors, *Advanced Topics in Bisimulation and Coinduction*, number 52 in Cambridge Tracts in Theoretical Computer Science, pages 233—289. Cambridge University Press, 2012.
- 21 Peter Thiemann. Partial Derivatives for Context-Free Languages - From μ -Regular Expressions to Pushdown Automata. In Javier Esparza and Andrzej S. Murawski, editors, *Proceedings of FOSSACS 2017*, volume 10203 of *LNCS*, pages 248–264, 2017. doi:10.1007/978-3-662-54458-7_15.
- 22 Chris Verhoef. A General Conservative Extension Theorem in Process Algebra. In Ernst-Rüdiger Olderog, editor, *Proceedings of PROCOMET'94*, volume A-56 of *IFIP Transactions*, pages 149–168. North-Holland, 1994.
- 23 Chris Verhoef. A Congruence Theorem for Structured Operational Semantics with Predicates and Negative Premises. *Nord. J. Comput.*, 2(2):274–302, 1995.
- 24 Jos L. M. Vrancken. The Algebra of Communicating Processes With Empty Process. *Theor. Comput. Sci.*, 177(2):287–328, 1997. doi:10.1016/S0304-3975(96)00250-2.

A Proofs of Lemmas 9 and 10

In this appendix we shall provide proofs for Lemmas 9 and 10, restated below as Lemmas 35 and 40. For the formulation of our arguments, it is convenient to associate behaviour to $\text{TSP}^i(\mathcal{A})$ -terms with variables. We assume an extended syntax in which a constant \bar{x} added for every variable x and include the following operational rule to the operational semantics presented in Figure 1:

$$\frac{}{x \xrightarrow{x} \bar{x}} .$$

11:18 Sequencing and Intermediate Acceptance

The resulting collection of operational rules is used to derive transitions of $\text{TSP}^i(\mathcal{A})$ -terms (with variables). A transition of a $\text{TSP}^i(\mathcal{A})$ -term t may then either result in another $\text{TSP}^i(\mathcal{A})$ -term, or, if it is due to the transition of a variable, it may result in a term in a syntax extended with the constant \bar{x} : For every variable x , we inductively define the set of $\text{TSP}^i(\mathcal{A}, \bar{x})$ -terms as follows:

1. the constant \bar{x} is a $\text{TSP}^i(\mathcal{A}, \bar{x})$ -term; and
2. if t_1 is a $\text{TSP}^i(\mathcal{A}, \bar{x})$ -term and t_2 is a $\text{TSP}^i(\mathcal{A})$ -term, then $t_1 ; t_2$ is a $\text{TSP}^i(\mathcal{A}, \bar{x})$ -term.

If t is a $\text{TSP}^i(\mathcal{A}, \bar{x})$ -term and p is a closed $\text{TSP}^i(\mathcal{A})$ -term, then by $t[\bar{x} := p]$ we denote the $\text{TSP}^i(\mathcal{A})$ -term obtained by replacing \bar{x} by p . While every variable can now “take a step”, we do not let this contribute to the width of a term, so $\text{width}(x) = 0$ for every variable x .

► **Lemma 29.** *Let t be a $\text{TSP}^i(\mathcal{A})$ -term.*

1. If $t \xrightarrow{a} t'$ for some action a , then t' is a $\text{TSP}^i(\mathcal{A})$ -term.
2. If $t \xrightarrow{x} t'$ for some variable x , then t' is a $\text{TSP}^i(\mathcal{A}, \bar{x})$ -term.

We would like to establish a relationship between transitions from t and transitions from $\sigma(t)$, where σ is a closed substitution. However, we cannot yet fully express that a transition originates from a substitution in a variable. For example, consider the $\text{TSP}^i(\mathcal{A})$ -term $t \equiv x ; y$ and the closed substitution σ , where $\sigma(x) = \mathbf{1}$ and $\sigma(y) = a.\mathbf{1}$. Clearly, $\sigma(t) = \mathbf{1} ; a.\mathbf{1}$ and hence $\sigma(t) \xrightarrow{a} \mathbf{1}$. However, we cannot express that this a -transition originates from the substitution in y , as $t \not\xrightarrow{y}$. To be able to express this, we define the following substitution.

► **Definition 30.** *Given a substitution σ and variable x , the substitution $\mu_{\sigma x}$ is defined as:*

$$\mu_{\sigma x}(y) = \begin{cases} y & \text{if } y = x \\ \sigma(y) & \text{otherwise.} \end{cases}$$

Referring to the example preceding Definition 30, note that $\mu_{\sigma y}(t) \xrightarrow{y} \bar{y}$ and $\sigma(y) \xrightarrow{a} \mathbf{1}$. We can establish several useful relationships between $\sigma(t)$ and $\mu_{\sigma x}(t)$.

► **Lemma 31.** *Let t be a $\text{TSP}^i(\mathcal{A})$ -term, σ a closed substitution, x a variable, p a closed $\text{TSP}^i(\mathcal{A})$ -term and a an action such that $\sigma(x) \xrightarrow{a} p$. Then:*

1. if $\sigma(t) \not\downarrow$, then $\mu_{\sigma x}(t) \not\downarrow$;
2. if $\sigma(t) \not\rightarrow$, then $\mu_{\sigma x}(t) \not\rightarrow$;
3. if $\sigma(t) \not\rightarrow$ and $\sigma(t) \downarrow$, then $\mu_{\sigma x}(t) \downarrow$.

In the following lemma it is proven that if t contains a subterm t_2 such that $\text{width}(\sigma(t_2)) > \text{width}(t_2)$, then one of the actions that can be executed by $\sigma(t_2)$ must come from a substitution in some variable x .

► **Lemma 32.** *Let t be a $\text{TSP}^i(\mathcal{A})$ -term and σ a closed substitution. If $\text{width}(\sigma(t)) > \text{width}(t)$, then there must exist an action a , closed $\text{TSP}^i(\mathcal{A})$ -terms p and p' , a $\text{TSP}^i(\mathcal{A}, \bar{x})$ -term t' , and a variable x such that $\sigma(t) \xrightarrow{a} p$, $\mu_{\sigma x}(t) \xrightarrow{x} t'$, $\sigma(x) \xrightarrow{a} p'$ and $p \equiv \sigma(t'[\bar{x} := p'])$.*

► **Lemma 33.** *Let t be a $\text{TSP}^i(\mathcal{A})$ -term, x a variable and σ a closed substitution. Then:*

1. if $t \downarrow$, then $\sigma(t) \downarrow$;
2. if $\sigma(t) \downarrow$, then $\vartheta_{(\sigma, x)}(t) \downarrow$;
3. if $\vartheta_{(\sigma, x)}(\mu_{\sigma x}(t)) \downarrow$, then $\vartheta_{(\sigma, x)}(t) \downarrow$.

Using these properties we show that given a term t , variable x and substitution σ as described above, $\vartheta_{(\sigma, x)}(t) \downarrow$ indeed holds.

► **Lemma 34.** *Let t be a $\text{TSP}^i(\mathcal{A})$ -term, let x be a variable, and let σ be a substitution. If there exist a $\text{TSP}^i(\mathcal{A}, \bar{x})$ -term t' and a closed $\text{TSP}^i(\mathcal{A})$ -term p such that $t \xrightarrow{x} t'$, $\sigma(x) \xrightarrow{a} p$ and $\sigma(t'[\bar{x} := p]) \downarrow$, then $\vartheta_{(\sigma, x)}(t) \downarrow$.*

By utilizing the results from Lemma 32 and Lemma 34, we show that given a $\text{TSP}^i(\mathcal{A})$ -term t and substitution σ , if $\Psi_n(\sigma(t))$ holds, then t must contain some variable x such that $\vartheta_{(\sigma, x)}(t) \downarrow$ and either $\Psi_n(\sigma(x))$ or $\sigma(x) \xrightarrow{\bar{b}}$.

► **Lemma 35.** *Let t be a $\text{TSP}^i(\mathcal{A})$ -term and let n be a natural number such that $\text{width}(t') < n$ for every subterm t' of t . If $\Psi_n(\sigma(t))$ for some closed substitution σ , then there is a variable x such that $\vartheta_{(\sigma, x)}(t) \downarrow$ and either $\Psi_n(\sigma(x))$ or $\sigma(x) \xrightarrow{\bar{b}}$.*

Proof. We proceed with induction on the structure of t .

- If $t \equiv \mathbf{0}$, $t \equiv \mathbf{1}$ or $t \equiv a.t'$ for some action a and $\text{TSP}^i(\mathcal{A})$ -term t' , then $\sigma(t)$ cannot have a summand of the form $t_1 ; t_2$, so $\Psi_n(\sigma(t))$ does not hold for any substitution σ . Hence the implication vacuously holds.
- Let $t \equiv y$ for some variable y , and suppose that $\Psi_n(\sigma(t))$ holds for some closed substitution σ . Then clearly from $\Psi_n(\sigma(t))$ it follows that $\Psi_n(\sigma(y))$. Furthermore, since $t \xrightarrow{y} \bar{y}$ and since $\sigma(\bar{y}[\bar{y} := \mathbf{1}]) \equiv \mathbf{1}$, we have that $\sigma(\bar{y}[\bar{y} := \mathbf{1}]) \downarrow$. Hence, by Lemma 34, we have that $\vartheta_{(\sigma, y)}(t) \downarrow$ and thus $x = y$.
- Let $t \equiv t_1 + t_2$ for some $\text{TSP}^i(\mathcal{A})$ -terms t_1 and t_2 . If $\Psi_n(\sigma(t))$, then either $\sigma(t_1)$ or $\sigma(t_2)$ must contain a summand p such that one of the three cases of the definition of Ψ_n applies. We proceed to consider the case that p is a summand of $\sigma(t_1)$; the proof in the case that p is a summand of $\sigma(t_2)$ proceeds analogously. Note that, since $p \Leftarrow (\tilde{a}.\mathbf{1} + \mathbf{1}) ; \sum_{i=1}^n (\tilde{b}.\mathbf{1} ; (\tilde{b}.\mathbf{1} + \mathbf{1})^i)$, we find that $\sigma(t_1) \xrightarrow{\tilde{a}} p'$ with $p' \Leftarrow \sum_{i=1}^n (\tilde{b}.\mathbf{1} ; (\tilde{b}.\mathbf{1} + \mathbf{1})^i)$. Moreover, since $\sigma(t_1)$ is a summand of $\sigma(t)$ and also $\sigma(t) \Leftarrow (\tilde{a}.\mathbf{1} + \mathbf{1}) ; \sum_{i=1}^n (\tilde{b}.\mathbf{1} ; (\tilde{b}.\mathbf{1} + \mathbf{1})^i)$, we find that $\sigma(t_1) \Leftarrow (\tilde{a}.\mathbf{1} + \mathbf{1}) ; \sum_{i=1}^n (\tilde{b}.\mathbf{1} ; (\tilde{b}.\mathbf{1} + \mathbf{1})^i)$, and hence $\Psi_n(\sigma(t_1))$. Since every subterm of t_1 is a subterm of t , we also have that $\text{width}(t') < n$ for every subterm t' of t_1 . Therefore, we may now apply the induction hypothesis to conclude that either $\Psi_n(\sigma(x))$ or $\sigma(x) \xrightarrow{\bar{b}}$, and $\vartheta_{(\sigma, x)}(t_1) \downarrow$; clearly, from the latter it follows that $\vartheta_{(\sigma, x)}(t) \downarrow$.
- Let $t \equiv t_1 ; t_2$ for some $\text{TSP}^i(\mathcal{A})$ -terms t_1 and t_2 , and suppose that $\Psi_n(\sigma(t))$. Then, considering the definition of Ψ_n , one of the following three cases must apply:
 1. If $\Phi_n(\sigma(t_1) ; \sigma(t_2))$, then $\sigma(t_1) \Leftarrow \tilde{a}.\mathbf{1} + \mathbf{1}$ and $\sigma(t_2) \Leftarrow \sum_{i=1}^n (\tilde{b}.\mathbf{1} ; (\tilde{b}.\mathbf{1} + \mathbf{1})^i)$. Then $\sigma(t_2) \xrightarrow{\tilde{b}} (\tilde{b}.\mathbf{1} + \mathbf{1})^i$ for all $1 \leq i \leq n$. Clearly, if $i \neq j$, then $(\tilde{b}.\mathbf{1} + \mathbf{1})^i \not\Leftarrow (\tilde{b}.\mathbf{1} + \mathbf{1})^j$, so $\text{width}(\sigma(t_2)) \geq n > \text{width}(t_2)$. It follows by Lemma 32 that there exist an action a , closed $\text{TSP}^i(\mathcal{A})$ -terms p and p' , a $\text{TSP}^i(\mathcal{A}, \bar{x})$ -term t' and a variable x such that $\sigma(t_2) \xrightarrow{a} p$, $\mu_{\sigma x}(t_2) \xrightarrow{x} t'$, $\sigma(x) \xrightarrow{a} p'$ and $p \equiv \sigma(t'[\bar{x} := p'])$. Clearly, we must have $a = \tilde{b}$ and $p \Leftarrow (\tilde{b}.\mathbf{1} + \mathbf{1})^i$ for some $1 \leq i \leq n$. To see that $\vartheta_{(\sigma, x)}(t) \downarrow$, note that, since $\sigma(t_1) \Leftarrow \tilde{a}.\mathbf{1} + \mathbf{1}$, we have that $\sigma(t_1) \downarrow$ and hence $\vartheta_{(\sigma, x)}(t_1) \downarrow$. Moreover, since $\sigma(t'[\bar{x} := p']) \Leftarrow (\tilde{b}.\mathbf{1} + \mathbf{1})^i$, we find that $\sigma(t'[\bar{x} := p']) \downarrow$, and hence, by Lemma 34, we get that $\vartheta_{(\sigma, x)}(\mu_{\sigma x}(t_2)) \downarrow$. Finally, by Lemma 33(3), we conclude that $\vartheta_{(\sigma, x)}(t_2) \downarrow$ and thus $\vartheta_{(\sigma, x)}(t) \downarrow$.
 2. If $\sigma(t_1) \Leftarrow \mathbf{1}$ and $\Psi_n(\sigma(t_2))$, then since every subterm of t_2 is a subterm of t we find that $\text{width}(t'_2) < n$ for all subterms t'_2 of t_2 . Hence, by the induction hypothesis, for some variable x we have that either $\Psi_n(\sigma(x))$ or $\sigma(x) \xrightarrow{\bar{b}}$ and, moreover, $\vartheta_{(\sigma, x)}(t_2) \downarrow$. From $\sigma(t_1) \Leftarrow \mathbf{1}$ it follows that $\sigma(t_1) \downarrow$, so, by Lemma 33(2), $\vartheta_{(\sigma, x)}(t_1) \downarrow$, and hence $\vartheta_{(\sigma, x)}(t) \downarrow$.

11:20 Sequencing and Intermediate Acceptance

3. If $\Psi_n(\sigma(t_1))$ and $\sigma(t_2) \Leftrightarrow \mathbf{1}$, then the proof that $\vartheta_{(\sigma,x)}(t) \downarrow$ is analogous to the previous case. \blacktriangleleft

We have established that if $\Psi_n(\sigma(t))$ holds for some substitution σ and $\text{TSP}^i(\mathcal{A})$ -term t such that $\text{width}(t') < n$ for each subterm t' of t , then t must confirm to certain properties. Now, for any term u such that $t \Leftrightarrow u$, these properties must be valid as well. Hence, it remains to show is that if u contains these properties, then $\Psi_n(\sigma(u))$ must hold as well. This is shown in Lemma 40. In order to prove this result, some useful properties are established in Lemma 36 to Lemma 39.

► **Lemma 36.** *Let p and q be closed $\text{TSP}^i(\mathcal{A})$ -terms and suppose that $p \Leftrightarrow q$. Then $\text{depth}(p) = \text{depth}(q)$.*

Proof. Assume that $p \Leftrightarrow q$ and, for the sake of contradiction, suppose that $\text{depth}(p) = n$ and $\text{depth}(q) = m$, for some $n > m$. Then, by definition, $p \longrightarrow^n p'$ and since $p \Leftrightarrow q$, $q \longrightarrow^n q'$, such that $p' \Leftrightarrow q'$. Clearly, since $n > m$, this contradicts $\text{depth}(q) = m$. Hence, we conclude $\text{depth}(p) = \text{depth}(q)$. \blacktriangleleft

► **Lemma 37.** *For all closed $\text{TSP}^i(\mathcal{A})$ -terms p_1 and p_2 , if $p_1 ; p_2 \Leftrightarrow \tilde{a}.\mathbf{1} ; \sum_{i=1}^n \tilde{b}.\tilde{b}.\mathbf{1} + \mathbf{1}^i$, then one of the following cases must hold:*

1. $p_1 \Leftrightarrow \mathbf{1}$ and $p_2 \Leftrightarrow \tilde{a}.\mathbf{1} ; \sum_{i=1}^n \tilde{b}.\tilde{b}.\mathbf{1} + \mathbf{1}^i$; or
2. $p_1 \Leftrightarrow \tilde{a}.\mathbf{1}$ and $p_2 \Leftrightarrow \sum_{i=1}^n \tilde{b}.\tilde{b}.\mathbf{1} + \mathbf{1}^i$; or
3. $p_1 \Leftrightarrow \tilde{a}.\mathbf{1} + \mathbf{1}$ and $p_2 \Leftrightarrow \tilde{a}.\mathbf{1} ; \sum_{i=1}^n \tilde{b}.\tilde{b}.\mathbf{1} + \mathbf{1}^i$; or
4. $p_1 \Leftrightarrow \tilde{a}.\mathbf{1} ; \sum_{i=1}^n \tilde{b}.\tilde{b}.\mathbf{1} + \mathbf{1}^i$ and $p_2 \Leftrightarrow \mathbf{1}$.

► **Lemma 38.** *For any $\text{TSP}^i(\mathcal{A})$ -term t , variable x and closed substitution σ , if $\sigma(t) \not\downarrow$ and $\vartheta_{(\sigma,x)}(t) \downarrow$, then t contains x .*

Proof. Let t be a $\text{TSP}^i(\mathcal{A})$ -term, x a variable and σ a closed substitution such that $\sigma(t) \not\downarrow$ and $\vartheta_{(\sigma,x)}(t) \downarrow$. Now suppose t does not contain x . Then, by the definition of $\vartheta_{(\sigma,x)}$, $\vartheta_{(\sigma,x)}(t) \equiv \sigma(t)$, which means that if $\sigma(t) \not\downarrow$ we should also have $\vartheta_{(\sigma,x)}(t) \not\downarrow$. Since this contradicts $\vartheta_{(\sigma,x)}(t) \downarrow$, we conclude t must contain x . \blacktriangleleft

► **Lemma 39.** *For any $\text{TSP}^i(\mathcal{A})$ -term t , variable x and closed substitution σ , if t contains x and $\sigma(x) \xrightarrow{a_1 \dots a_n} p$ for some sequence of actions $a_1 \dots a_n$ and closed $\text{TSP}^i(\mathcal{A})$ -term p , then*

1. either $\sigma(t) \longrightarrow^* p' \xrightarrow{a_1 \dots a_n} p''$, for some p' and p'' ,
2. or $\sigma(t) \longrightarrow^* p'$, for some p' such that $p' \Leftrightarrow \mathbf{0}$.

Proof. Let t be a $\text{TSP}^i(\mathcal{A})$ -term, x a variable and σ a closed substitution such that t contains x and $\sigma(x) \xrightarrow{a_1 \dots a_n} p$ for some sequence of actions $a_1 \dots a_n$ and closed $\text{TSP}^i(\mathcal{A})$ -term p . It can then be proved with induction on the structure of t that one of the two cases of the lemma must hold. \blacktriangleleft

► **Lemma 40.** *For any $\text{TSP}^i(\mathcal{A})$ -term t , variable x and closed substitution σ , if $\sigma(t) \Leftrightarrow (\tilde{a}.\mathbf{1} + \mathbf{1}) ; \sum_{i=1}^n (\tilde{b}.\mathbf{1} ; (\tilde{b}.\mathbf{1} + \mathbf{1})^i)$, $\vartheta_{(\sigma,x)}(t) \downarrow$ and either $\Psi_n(\sigma(x))$ or $\sigma(x) \xrightarrow{\tilde{b}}$, then $\Psi_n(\sigma(t))$.*

Proof. Let t be a $\text{TSP}^i(\mathcal{A})$ -term, x a variable and let σ be closed substitution such that $\sigma(t) \Leftrightarrow (\tilde{a}.\mathbf{1} + \mathbf{1}) ; \sum_{i=1}^n (\tilde{b}.\mathbf{1} ; (\tilde{b}.\mathbf{1} + \mathbf{1})^i)$, $\vartheta_{(\sigma,x)}(t) \downarrow$ and either $\Psi_n(\sigma(x))$ or $\sigma(x) \xrightarrow{\tilde{b}}$. We prove by induction on the structure of t that $\Psi_n(\sigma(t))$ must hold.

- If $t \equiv \mathbf{0}$ or $t \equiv \mathbf{1}$, then $\sigma(t) \not\downarrow (\tilde{a}.\mathbf{1} + \mathbf{1}) ; \sum_{i=1}^n (\tilde{b}.\mathbf{1} ; (\tilde{b}.\mathbf{1} + \mathbf{1})^i)$, hence, the implication vacuously holds.

- If $t \equiv a.t'$, then $\vartheta_{(\sigma,x)}(t) \not\downarrow$ which contradicts $\vartheta_{(\sigma,x)}(t) \downarrow$, hence, the implication vacuously holds.
- If $t \equiv y$ for some variable y , then since $\sigma(t) \Leftrightarrow (\tilde{a}.\mathbf{1} + \mathbf{1}); \sum_{i=1}^n (\tilde{b}.\mathbf{1}; (\tilde{b}.\mathbf{1} + \mathbf{1})^i)$, we must have $\sigma(t) \not\downarrow$. Moreover, since $\vartheta_{(\sigma,x)}(t) \downarrow$, by Lemma 38, $\sigma(t)$ contains x , thus we must have $y \equiv x$. Now suppose $\sigma(x) \xrightarrow{\tilde{b}}$, then $\sigma(t) \xrightarrow{\tilde{b}}$, contradicting $\sigma(t) \Leftrightarrow (\tilde{a}.\mathbf{1} + \mathbf{1}); \sum_{i=1}^n (\tilde{b}.\mathbf{1}; (\tilde{b}.\mathbf{1} + \mathbf{1})^i)$. Hence, it must be the case that $\Psi_n(\sigma(x))$ holds and thus also $\Psi_n(\sigma(t))$ holds.
- Suppose $t \equiv t_1 + t_2$ and suppose that the lemma holds for t_1 and t_2 (IH). Since $\sigma(t) \Leftrightarrow (\tilde{a}.\mathbf{1} + \mathbf{1}); \sum_{i=1}^n (\tilde{b}.\mathbf{1}; (\tilde{b}.\mathbf{1} + \mathbf{1})^i)$, we have $\sigma(t) \not\downarrow$ and thus both $\sigma(t_1) \not\downarrow$ and $\sigma(t_2) \not\downarrow$. Moreover, since $\vartheta_{(\sigma,x)}(t) \downarrow$ either $\vartheta_{(\sigma,x)}(t_1) \downarrow$ or $\vartheta_{(\sigma,x)}(t_2) \downarrow$. Without loss of generality assume $\vartheta_{(\sigma,x)}(t_1) \downarrow$. Then, by Lemma 38, t_1 must contain x . Since either $\sigma(x) \xrightarrow{\tilde{b}}$ or $\Psi_n(\sigma(x))$ and thus $\sigma(x) \xrightarrow{\tilde{a}}$, by Lemma 39, either $\sigma(t_1) \xrightarrow{*} q_1 \xrightarrow{a}$ for some action a and closed $\text{TSP}^i(\mathcal{A})$ -term q_1 , or $\sigma(t_1) \xrightarrow{*} q_2$ for some closed $\text{TSP}^i(\mathcal{A})$ -term q_2 such that $q_2 \Leftrightarrow \mathbf{0}$. The second case clearly contradicts $\sigma(t) \Leftrightarrow (\tilde{a}.\mathbf{1} + \mathbf{1}); \sum_{i=1}^n (\tilde{b}.\mathbf{1}; (\tilde{b}.\mathbf{1} + \mathbf{1})^i)$. Hence, it must be the case that $\sigma(t_1) \xrightarrow{*} q_1 \xrightarrow{\tilde{a}}$. Since $\sigma(t_1)$ is able to execute an action and $\sigma(t_1)$ is a summand of $\sigma(t)$, we must have $\sigma(t_1) \xrightarrow{a} p$ such that $p \Leftrightarrow \sum_{i=1}^n (\tilde{b}.\mathbf{1}; (\tilde{b}.\mathbf{1} + \mathbf{1})^i)$. Hence, we must have $\sigma(t_1) \Leftrightarrow (\tilde{a}.\mathbf{1} + \mathbf{1}); \sum_{i=1}^n (\tilde{b}.\mathbf{1}; (\tilde{b}.\mathbf{1} + \mathbf{1})^i)$, and thus by the induction hypothesis we conclude $\Psi_n(\sigma(t_1))$, and since $\sigma(t_1)$ is a summand of $\sigma(t)$ also $\Psi_n(\sigma(t))$.
- Suppose $t \equiv t_1 ; t_2$ and suppose that the lemma holds for t_1 and t_2 (IH). Since $\sigma(t) = \sigma(t_1 ; t_2) = \sigma(t_1) ; \sigma(t_2)$, we have $\sigma(t_1) ; \sigma(t_2) \Leftrightarrow (\tilde{a}.\mathbf{1} + \mathbf{1}); \sum_{i=1}^n (\tilde{b}.\mathbf{1}; (\tilde{b}.\mathbf{1} + \mathbf{1})^i)$ and thus by Lemma 37, one of the following cases must hold:
 1. If $\sigma(t_1) \Leftrightarrow \mathbf{1}$ and $\sigma(t_2) \Leftrightarrow (\tilde{a}.\mathbf{1} + \mathbf{1}); \sum_{i=1}^n (\tilde{b}.\mathbf{1}; (\tilde{b}.\mathbf{1} + \mathbf{1})^i)$, then, by the induction hypothesis, $\Psi_n(\sigma(t_2))$. Moreover, since $\sigma(t_1) \Leftrightarrow \mathbf{1}$, by case 2 of Ψ_n we conclude $\Psi_n(\sigma(t))$.
 2. If $\sigma(t_1) \Leftrightarrow \tilde{a}.\mathbf{1}$ and $\sigma(t_2) \Leftrightarrow \sum_{i=1}^n (\tilde{b}.\mathbf{1}; (\tilde{b}.\mathbf{1} + \mathbf{1})^i)$, then $\sigma(t_1) \not\downarrow$. Moreover, since $\vartheta_{(\sigma,x)}(t) \downarrow$ we must have $\vartheta_{(\sigma,x)}(t_1) \downarrow$, and thus by Lemma 38, t_1 must contain x . We distinguish two cases.
 - If $\Psi_n(\sigma(x))$, then $\sigma(x) \Leftrightarrow (\tilde{a}.\mathbf{1} + \mathbf{1}); \sum_{i=1}^n (\tilde{b}.\mathbf{1}; (\tilde{b}.\mathbf{1} + \mathbf{1})^i)$ and, by Lemma 39, either $\sigma(t_1) \xrightarrow{*} q_1 \xrightarrow{\tilde{a}} q'_1 \xrightarrow{\tilde{b}} q''_1$ for some closed $\text{TSP}^i(\mathcal{A})$ -terms q_1, q'_1 and q''_1 , or $\sigma(t_1) \xrightarrow{*} q_2$ for some closed $\text{TSP}^i(\mathcal{A})$ -term q_2 such that $q_2 \Leftrightarrow \mathbf{0}$. Both cases clearly contradict $\sigma(t_1) \Leftrightarrow \tilde{a}.\mathbf{1}$.
 - If $\sigma(x) \xrightarrow{\tilde{b}}$, then, by Lemma 39, either $\sigma(t_1) \xrightarrow{*} q_1 \xrightarrow{\tilde{b}}$ for some closed $\text{TSP}^i(\mathcal{A})$ -term q_1 , or $\sigma(t_1) \xrightarrow{*} q_2$ for some closed $\text{TSP}^i(\mathcal{A})$ -term q_2 such that $q_2 \Leftrightarrow \mathbf{0}$. Again, both cases contradict $\sigma(t_1) \Leftrightarrow \tilde{a}.\mathbf{1}$, hence the case where $\sigma(t_1) \Leftrightarrow \tilde{a}.\mathbf{1}$ and $\sigma(t_2) \Leftrightarrow (\tilde{a}.\mathbf{1} + \mathbf{1}); \sum_{i=1}^n (\tilde{b}.\mathbf{1}; (\tilde{b}.\mathbf{1} + \mathbf{1})^i)$ can never occur.
 3. If $\sigma(t_1) \Leftrightarrow \tilde{a}.\mathbf{1} + \mathbf{1}$ and $\sigma(t_2) \Leftrightarrow \sum_{i=1}^n (\tilde{b}.\mathbf{1}; (\tilde{b}.\mathbf{1} + \mathbf{1})^i)$, then clearly $\Phi_n(\sigma(t_1) ; \sigma(t_2))$ and thus, by case 1 of Ψ_n we conclude $\Psi_n(\sigma(t))$.
 4. If $\sigma(t_1) \Leftrightarrow (\tilde{a}.\mathbf{1} + \mathbf{1}); \sum_{i=1}^n (\tilde{b}.\mathbf{1}; (\tilde{b}.\mathbf{1} + \mathbf{1})^i)$ and $\sigma(t_2) \Leftrightarrow \mathbf{1}$, then, by the induction hypothesis, $\Psi_n(\sigma(t_1))$. Moreover, since $\sigma(t_2) \Leftrightarrow \mathbf{1}$, by case 3 of Ψ_n we conclude $\Psi_n(\sigma(t))$. ◀

B Proof of Proposition 21

The three properties of Proposition 21 are proved below as Lemmas 45, 46 and 47. Throughout this appendix it will be assumed that Δ is in AIGNF.

Let us first establish that then all Δ -reachable sequences are acceptance irredundant.

11:22 Sequencing and Intermediate Acceptance

► **Lemma 41.** For every sequence $\alpha \in \overline{\mathcal{P}}_{\downarrow}^*$, if $\alpha \xrightarrow{a} \alpha'$, then $\alpha' \in \overline{\mathcal{P}}_{\downarrow}^*$.

► **Lemma 42.** For every acceptance irredundant sequence $\alpha\beta$, we have that either $\alpha \in \overline{\mathcal{P}}_{\downarrow}^*$ and $\beta \in \overline{\mathcal{P}}_{\downarrow}^* \overline{\mathcal{P}}_{\downarrow} \mathcal{P}_{\downarrow}^*$, or $\alpha \in \overline{\mathcal{P}}_{\downarrow}^* \mathcal{P}_{\downarrow} \mathcal{P}_{\downarrow}^* \cup \mathcal{P}_{\downarrow}^*$ and $\beta \in \mathcal{P}_{\downarrow}^*$.

Using the previous lemma, we show that acceptance irredundant sequences maintain their shape when executing an action.

► **Lemma 43.** If α is acceptance irredundant and $\alpha \xrightarrow{a} \alpha'$, then α' is acceptance irredundant.

► **Corollary 44.** If Δ is in AIGNF, then all Δ -reachable sequences are acceptance irredundant.

► **Lemma 45.** Suppose that Δ is in AIGNF and let $\alpha, \beta \in \mathcal{P}^*$. If $\alpha\beta$ is acceptance irredundant and α is unnormed, then $\alpha\beta \leftrightarrow \alpha$.

Proof. We prove that the relation

$$R = \{(\alpha, \alpha\beta) \mid \alpha\beta \text{ is acceptance irredundant and } \alpha \text{ is unnormed}\}$$

is a bisimulation relation.

If $\alpha \xrightarrow{a} \alpha'$, then $\alpha\beta \xrightarrow{a} \alpha'\beta$ and since α is unnormed, so is α' . Moreover, since $\alpha\beta$ is acceptance irredundant, we have by Lemma 43 that $\alpha'\beta$ is acceptance irredundant and hence $(\alpha', \alpha'\beta) \in R$. Furthermore, if $\alpha \downarrow$, then $\alpha \in \mathcal{P}_{\downarrow}^*$, hence, since $\alpha\beta$ is acceptance irredundant, we have $\beta \in \mathcal{P}_{\downarrow}^*$ and thus $\alpha\beta \downarrow$. A symmetric argument applies for the cases where $\alpha\beta \xrightarrow{a} \alpha'\beta$ and $\alpha\beta \downarrow$. ◀

► **Lemma 46.** Suppose that Δ is in AIGNF. Then for all acceptance irredundant sequences α we have $|\alpha| \leq n(\alpha)$.

Proof. If α contains an unnormed process identifier, then clearly α is unnormed and hence $|\alpha| \leq n(\alpha) = \infty$. So, suppose that α is normed. Then all process identifiers in α are normed and must have a defining equation of the shape $\sum_{i=1}^n a_i.\alpha_i(+\mathbf{1})$ for some $n \in \mathbb{N}^+$ with α_i acceptance irredundant. Since every variable must be able to execute at least one action, the norm of each variable must be greater or equal than 1. Hence, in this case also $|\alpha| \leq n(\alpha)$. ◀

► **Lemma 47.** Suppose that Δ is in AIGNF and let $\alpha\gamma, \beta\gamma \in \mathcal{P}^*$ be acceptance irredundant. If α, β and γ are normed, then $\alpha\gamma \leftrightarrow \beta\gamma$ implies $\alpha \leftrightarrow \beta$.

Proof. It suffices to prove that

$$R = \{(\alpha, \beta) \mid \exists \gamma. \alpha\gamma \leftrightarrow \beta\gamma \text{ and } \alpha\gamma, \beta\gamma \text{ are acceptance irredundant}\}$$

is a bisimulation relation.

Suppose $\alpha \xrightarrow{a} \alpha'$, then $\beta\gamma \xrightarrow{a} \delta$ such that $\delta \leftrightarrow \alpha'\gamma$. We distinguish two cases.

- If $\beta \xrightarrow{a} \beta'$ and $\delta = \beta'\gamma$, then since $\alpha\gamma$ and $\beta\gamma$ are acceptance irredundant, by Lemma 43, $\alpha'\gamma$ and $\beta'\gamma$ are acceptance irredundant and therefore $(\alpha', \beta') \in R$.
- If $\beta \downarrow$, $\beta \dashv$, $\gamma \xrightarrow{a} \gamma'$ and $\delta = \gamma'$, then $n(\beta\gamma) = n(\gamma) < n(\alpha\gamma)$, contradicting $\alpha\gamma \leftrightarrow \beta\gamma$. Hence, in this case the implication vacuously holds.

Moreover, if $\alpha \downarrow$, then $\alpha \in \mathcal{P}_{\downarrow}^*$ and since $\alpha\gamma$ is acceptance irredundant also $\gamma \in \mathcal{P}_{\downarrow}^*$. Hence, we have $\alpha\gamma \downarrow$ and thus $\beta\gamma \downarrow$ and $\beta \downarrow$. A symmetric argument applies for the cases where $\beta \xrightarrow{a} \beta'$ and $\beta \downarrow$. ◀

On Terminal Coalgebras Derived from Initial Algebras

Jiří Adámek

Department of Mathematics, Faculty of Electrical Engineering,
Czech Technical University in Prague, Czech Republic
j.adamek@tu-bs.de

Abstract

A number of important set functors have countable initial algebras, but terminal coalgebras are uncountable or even non-existent. We prove that the countable cardinality is an anomaly: every set functor with an initial algebra of a finite or uncountable regular cardinality has a terminal coalgebra of the same cardinality.

We also present a number of categories that are algebraically complete and cocomplete, i.e., every endofunctor has an initial algebra and a terminal coalgebra.

Finally, for finitary set functors we prove that the initial algebra μF and terminal coalgebra νF carry a canonical ultrametric with the joint Cauchy completion. And the algebra structure of μF determines, by extending its inverse continuously, the coalgebra structure of νF .

2012 ACM Subject Classification Theory of computation → Categorical semantics

Keywords and phrases terminal coalgebras, initial algebras, algebraically complete category, finitary functor, fixed points of functors

Digital Object Identifier 10.4230/LIPIcs.CALCO.2019.12

Funding *Jiří Adámek*: Supported by the Grant Agency of the Czech Republic under the grant 19-00902S.

Acknowledgements The referees helped improving the presentation of this paper by numerous valuable suggestions.

1 Introduction

Initial algebras for endofunctors are important in formal semantics and the theory of recursive domain equations. Further, for state based systems represented as coalgebras, Rutten [11] demonstrated that the terminal coalgebra formalizes behavior of states. If we work in the category of *cpo*'s as our base category, and if the given endofunctor is locally continuous, then Smyth and Plotkin proved in [12] that the initial algebra coincides with the terminal coalgebra. That is, the underlying objects are equal, and the structure maps are inverse to each other.

Is there a connection between initial algebras μF and terminal coalgebras νF for set functors F , too? In the case where F preserves limits of ω^{op} -chains, νF carries a canonical structure of a metric space and, whenever $F\emptyset \neq \emptyset$, this is the Cauchy completion of μF as its subspace, as proved by Barr [8]. But what can we say about general set functors? There are cases where μF is countable and νF is uncountable (e.g. $F X = A \times X + 1$, with $\mu F = A^*$ and $\nu F = A^\infty$) or νF does not exist:

► **Example 1** (see [4]). The following set functor F has a countable initial algebra but no terminal coalgebra:

$$F X = \{M \subseteq X ; \text{card } M \neq \aleph_0\}.$$



© Jiří Adámek;

licensed under Creative Commons License CC-BY

8th Conference on Algebra and Coalgebra in Computer Science (CALCO 2019).

Editors: Markus Roggenbach and Ana Sokolova; Article No. 12; pp. 12:1–12:21

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

12:2 On Terminal Coalgebras Derived from Initial Algebras

To a function $f: X \rightarrow Y$ it assigns the function Ff turning $M \subseteq X$ to $f[M]$ if f restricted to M is monic or M is finite, else to \emptyset . Its initial algebra is that of the finite power-set functor (consisting of all hereditarily finite sets).

We are going to prove that the cardinal \aleph_0 is the only exception: whenever a set functor has a nonempty initial algebra of a finite or uncountable regular cardinality, then it has a terminal coalgebra of the same cardinality. See the Terminal-Coalgebra Theorem in Section 4. We also prove that the existence of a fixed point $FX \simeq X$ of an uncountable regular cardinality implies that the set functor F has a terminal coalgebra. This corresponds well with the result of [15] that every set functor with a fixed point has an initial algebra.

On the way to proving these results we present a number of categories that are algebraically complete and cocomplete. The concept of algebraic completeness, due to Freyd [9], means that every endofunctor has an initial algebra. But Freyd did not present any examples. It may seem that there are no “natural” examples since, as proved in [4], an algebraically complete category cannot be complete, unless it is equivalent to a preordered class. However, we prove that for every uncountable, regular cardinal λ the category $\mathbf{Set}_{\leq \lambda}$ of sets of cardinality at most λ is algebraically complete and cocomplete. That is, every endofunctor F has both μF and νF . For $\lambda > \aleph_1$ (the first uncountable cardinal) the category $\mathbf{Nom}_{\leq \lambda}$ of nominal sets of cardinality at most λ is also algebraically complete and cocomplete. Analogously, the category $K\text{-}\mathbf{Vec}_{\leq \lambda}$ of vector spaces of dimension at most λ , for any field K with $|K| < \lambda$, is algebraically complete and cocomplete. Finally, if G is a group, consider the category $G\text{-}\mathbf{Set}$ of sets with an action of G . For every group with $2^{|G|} < \lambda$ the category $G\text{-}\mathbf{Set}_{\leq \lambda}$ of G -sets of cardinality at most λ is algebraically complete and cocomplete. These results require assuming the Generalized Continuum Hypothesis.

Returning to metric structures on terminal coalgebras, we prove that for finitary set functors F with $F\emptyset \neq \emptyset$ the initial algebra and terminal coalgebra carry a canonical ultrametric such that the Cauchy completions of μF and νF coincide. And the coalgebra structure of νF is determined by the algebra structure ι of μF : it is the unique continuous extension of ι^{-1} to νF . This complements the above result of Barr [8].

2 Algebraically Cocomplete Categories

For a number of categories \mathcal{K} we prove that the full subcategory $\mathcal{K}_{\leq \lambda}$ on objects of power at most λ is algebraically cocomplete. Power is a cardinal we introduce as follows:

► **Definition 2.** An object is called *connected* if it is non-initial and is not a coproduct of two non-initial objects. An object is said to have *power* λ if it is a coproduct of λ connected objects, but not of less than λ ones.

► **Example 3.** In \mathbf{Set} , connected objects are the singleton sets, and power of a set X is its cardinality $|X|$. In the category $K\text{-}\mathbf{Vec}$ of vector spaces over a field K the connected spaces are those of dimension one, and power means dimension. In the category \mathbf{Set}^S of many-sorted sets the connected objects are those with precisely one element (in all sorts together), and the power of $X = (X_s)_{s \in S}$ is simply $|\prod_{s \in S} X_s|$. A nominal set is connected in the category \mathbf{Nom} of nominal sets and equivariant maps iff it consists of a single orbit.

► **Definition 4.** A category \mathcal{K} is said to have *width* $w(\mathcal{K})$ if it has coproducts, every object is a coproduct of connected objects, and $w(\mathcal{K})$ is the smallest cardinal such that

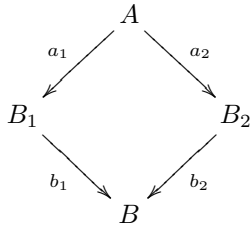
- (a) \mathcal{K} has at most $w(\mathcal{K})$ connected objects up to isomorphism, and
- (b) given an object K of power $\alpha \geq w(\mathcal{K})$, all quotients of K have power at most α , and there exist at most α morphisms from a connected object to K .

► **Example 5.**

- (1) **Set** has width 1. More generally, \mathbf{Set}^S has width $|S|$. Indeed, in Example 3 we have seen that the number of connected objects up to isomorphism is $|S|$, and (b) clearly holds.
- (2) $K\text{-Vec}$ has width $|K| + \aleph_0$. Indeed, the only connected object, up to isomorphism, is K . For a space X of dimension α the number of morphisms from K to X is $|X|$. If K is infinite, then $\alpha \geq |K|$ implies $|X| = \alpha$ (and $|K| = |K| + \aleph_0$). For K finite, the least cardinal λ such that $|X| \leq \alpha$ holds for every α -dimensional spaces X with $\alpha \geq \lambda$ is \aleph_0 ($= |K| + \aleph_0$).
- (3) The category **Nom** of nominal sets has width \aleph_0 .
- (4) For every nontrivial finite group G the category $G\text{-Set}$ of sets with an action of the group has width \aleph_0 . For infinite groups the width of $G\text{-Set}$ is at most λ if $2^{|G|} \leq \lambda$. For the proof of (3) and (4) see the Appendix.

We now present some technical results serving for the proof of Theorem 13 below. The following lemma is based on ideas of Trnková [14].

► **Lemma 6.** *Let a commutative square*



be given in a category \mathcal{A} . This is an absolute pullback, i.e., a pullback preserved by all functors with domain \mathcal{A} , provided that (1) b_1 and b_2 are split monomorphisms, and (2) there exist morphisms $\bar{b}_1 : B \rightarrow B_1$ and $\bar{a}_2 : B_2 \rightarrow A$ satisfying

$$\bar{b}_1 b_1 = \text{id}, \quad \bar{a}_2 a_2 = \text{id}, \quad \text{and} \quad a_1 \bar{a}_2 = \bar{b}_1 b_2. \tag{2.1}$$

Proof. The given square is a pullback since given a commutative square

$$b_1 c_1 = b_2 c_2 \quad \text{for} \quad c_i : C \rightarrow B_i$$

there exists a unique c with $c_i = a_i \cdot c$ ($i = 1, 2$). Uniqueness is clear since a_2 is split monic. Put $c = \bar{a}_2 \cdot c_2$. Then $c_1 = a_1 c$ follows from b_1 being monic:

$$\begin{aligned} b_1 c_1 &= b_1 \bar{b}_1 b_1 c_1 & \bar{b}_1 b_1 &= \text{id} \\ &= b_1 \bar{b}_1 b_2 c_2 & b_1 c_1 &= b_2 c_2 \\ &= b_1 a_1 \bar{a}_2 c_2 & \bar{b}_1 b_2 &= a_1 \bar{a}_2 \\ &= b_1 a_1 c & c &= \bar{a}_2 c_2 \end{aligned}$$

And $c_2 = a_2 c$ follows from b_2 being monic:

$$\begin{aligned} b_2 c_2 &= b_1 c_1 & c_1 &= a_1 c \\ &= b_1 a_1 c & b_1 a_1 &= b_2 a_2 \\ &= b_2 a_2 c \end{aligned}$$

For every functor F with domain \mathcal{A} the image of the given square satisfies the analogous conditions: Fb_1 and Fb_2 are split monomorphisms and $F\bar{b}_1, F\bar{a}_2$ verify (2.1). Thus, the image is an (absolute) pullback, too. ◀

12:4 On Terminal Coalgebras Derived from Initial Algebras

► **Corollary 7** (See [14]). *Every set functor preserves nonempty finite intersections.*

Indeed if A in Lemma 6 is nonempty, choose an element $t \in A$ and define \bar{b}_1 and \bar{a}_2 by

$$\bar{b}_1(x) = \begin{cases} y & \text{if } b_1(y) = x \\ a_1(t) & \text{if } x \notin b_1[B_1] \end{cases}$$

and

$$\bar{a}_2(x) = \begin{cases} y & \text{if } a_2(y) = x \\ t & \text{if } x \notin a_2[A] \end{cases}$$

It is easy to see that (2.1) holds.

► **Remark 8.**

- (a) We recall that for an infinite cardinal λ the *cofinality* is the smallest cardinal μ such that λ is a join of a μ -chain of smaller cardinals. And λ is *regular* if it is equal to its cofinality. The first non-regular cardinal is $\aleph_\omega = \bigvee_{n < \omega} \aleph_n$.
- (b) For a set X of infinite cardinality λ a collection of subsets of cardinality λ is called *almost disjoint* if the intersection of arbitrary two distinct members has cardinality smaller than λ .

Tarski [13] proved that for every set X of infinite regular cardinality λ there exists an almost disjoint collection $Y_i \subseteq X$ ($i \in I$) with $|I| > \lambda$. The argument is quite simple. Using the Maximality Principle (also known as Zorn's Lemma), we see that a maximum almost disjoint collection exists on X . Assuming that it has at most λ members, we derive a contradiction. We can index that collection by ordinals $i < \lambda$. Given an almost disjoint collection $(Y_i)_{i < \lambda}$, the following sets $Z_i = Y_i - \bigcup_{j < i} Y_j$ for $i < \lambda$ are clearly pairwise disjoint and, since λ is regular, they have cardinality λ . We can find a choice set $Z^* \subseteq X$. From $|Z^* \cap Z_i| = 1$ it follows that $|Z^* \cap Y_i| < \lambda$ for every $i < \lambda$, thus, we can add Z^* to the given collection. This contradicts the maximality.

- (c) Given an element $t \in X$ there exists a maximum almost disjoint collection Y_i , $i \in I$, with $t \in Y_i$ for all $i \in I$. Indeed, take any maximum collection $(Y_i)_{i \in I}$ and use $Y_i \cup \{t\}$ instead of Y_i (for $i \in I$).

► **Notation 9.** Let \mathcal{K} be a category of width $w(\mathcal{K})$. For every infinite cardinal $\lambda > w(\mathcal{K})$ we denote by $\mathcal{K}_{\leq \lambda}$ the full subcategory of \mathcal{K} on objects of power at most λ .

Our main technical tool is the following

► **Proposition 10.** *Let F be an endofunctor of $\mathcal{K}_{\leq \lambda}$ and $X = \prod_{i \in I} X_i$ an object of \mathcal{K} with all X_i connected and $|I| = \lambda$. Every morphism $b: B \rightarrow FX$, with B of power less than λ , factorizes through Fc for a coproduct injection $c: C \rightarrow X$ where $C = \prod_{j \in J} X_j$ and $|J| < \lambda$.*

The proof can be found in the Appendix.

► **Proposition 11.** *Let λ be an uncountable regular cardinal. Every coalgebra for an endofunctor of $\mathcal{K}_{\leq \lambda}$ is a colimit of a λ -filtered diagram of coalgebras on objects of powers smaller than λ .*

Proof. Let $\beta : B \rightarrow FB$ be a coalgebra. Express $B = \coprod_{i \in I} B_i$ where B_i are connected and $|I| = \lambda$ (the case $|I| < \lambda$ is trivial). For every set $J \subseteq I$ with $|J| < \lambda$ we are going to prove that there exists a set $J \subseteq J' \subseteq I$ with $|J'| < \lambda$ such that the summand

$$u_{J'} : B_{J'} = \coprod_{i \in J'} B_i \rightarrow B$$

carries a subcoalgebra. That is, there exists $\beta_{J'} : B_{J'} \rightarrow FB_{J'}$ for which $u_{J'}$ is a homomorphism. This proves the proposition: the diagram of all subcoalgebras of (B, β) on summands of less than λ components is clearly λ -filtered. And its canonical colimit is (B, β) . This follows from the fact that colimits of coalgebras are formed on the level of the underlying category.

For every set $J \subseteq I$ with $|J| < \lambda$ we are to present a set $J \subseteq J' \subseteq I$ with $|J'| < \lambda$ such that $\beta_{J'}$ exists. Put $J' = \bigcup_{n < \omega} J_n$ for the following ω -chain of sets $J_n \subseteq I$ with $|J_n| < \lambda$. First, $J_0 = J$. Given J_n , define J_{n+1} as follows. For every subset $L \subseteq J$ denote by $u_L : \coprod_{i \in L} B_i \rightarrow B$ the coproduct injection. Given $j \in J_n$, apply Proposition 10 to $X = B$ and

$$b = B_j \xrightarrow{u_{\{j\}}} B \xrightarrow{\beta} FB.$$

There exists a set $L(j) \subseteq J$ with $|L(j)| < \lambda$ such that the morphism $\beta \cdot u_{\{j\}}$ factorizes through $Fu_{L(j)}$. Consequently, for the set $L_n = \bigcup_{j \in J_n} L(j)$ we see that $\beta \cdot u_{J_n}$ factorizes through Fu_{L_n} . That is, there exists a morphism $\beta^n : \coprod_{i \in J_n} B_i \rightarrow \coprod_{i \in L_n} B_i$ with $Fu_{L_n} \cdot \beta^n = \beta \cdot u_{J_n}$.

Define $J_{n+1} = J_n \cup L_n$, then $|J_{n+1}| < \lambda + \prod_{j \in J} \lambda \leq \lambda + \lambda^2 = \lambda$.

Thus, for the union $J' = \bigcup J_n$ we get $|J'| < \lambda$ because λ is uncountable and regular, therefore $|\prod_{n < \omega} J_n| < \lambda$. And $u_{J'}$ carries the following subcoalgebra $\beta_{J'} : \prod_{j \in J'} B_j \rightarrow F(\prod_{j \in J'} B_j)$:

Given $j \in J'$ let n be the least number with $j \in J_n$. Denote by $w : B_j \rightarrow \prod_{i \in J_n} B_i$ and $v : \prod_{i \in L_n} B_i \rightarrow \prod_{j \in J'} B_j$ the coproduct injections. Then the j -th component of β' is the following composite

$$B_j \xrightarrow{w} \prod_{i \in J_n} B_i \xrightarrow{\beta^n} F(\prod_{i \in L_n} B_i) \xrightarrow{Fv} F(\prod_{i \in J'} B_i)$$

To prove that the following square

$$\begin{array}{ccc} \prod_{j \in J'} B_j & \xrightarrow{\beta_{J'}} & F(\prod_{j \in J'} B_j) \\ u_{J'} \downarrow & & \downarrow Fu_{J'} \\ \prod_{i \in I} B_i & \xrightarrow{\beta} & F(\prod_{i \in I} B_i) \end{array}$$

commutes, consider the components for $j \in J'$ separately. The upper passage yields, since $u_{J'} \cdot v = u_{L_n} : \prod_{i \in L_n} B_i \rightarrow \prod_{i \in I} B_i$, the result

$$Fu_{J'} \cdot (Fv \cdot \beta^n \cdot w) = Fu_{L_n} \cdot \beta^n \cdot w = \beta \cdot u_{J_n} \cdot w.$$

The lower passage yields the same result. ◀

12:6 On Terminal Coalgebras Derived from Initial Algebras

► Remark 12.

- (a) Every ordinal α is considered as the set of all smaller ordinals. In particular \aleph_0 is the set of all natural numbers, and \aleph_1 the set of all countable ordinals.
- (b) For cardinals λ and μ the power λ^μ is cardinality of the set of all functions from μ to λ .
- (c) If an infinite cardinal λ has cofinality μ , then $\lambda^\mu > \lambda$, see [10], Corollary 1.6.4.
- (d) Recall the *General Continuum Hypothesis (GCH)* which states that for every infinite cardinal λ the successor cardinal is 2^λ .

Under GCH every infinite regular cardinal λ fulfils $\lambda^\mu = \lambda$ for all cardinals $1 \leq \mu < \lambda$. See Theorem 1.6.17 in [10].

► **Theorem 13.** *Assume GCH. If \mathcal{K} is a cocomplete and cowellpowered category of width $w(\mathcal{K})$, then $\mathcal{K}_{\leq \lambda}$ is algebraically cocomplete for all uncountable regular cardinals $\lambda > w(\mathcal{K})$.*

Proof. Let F be an endofunctor of $\mathcal{K}_{\leq \lambda}$. Form a collection $a_i: A_i \rightarrow FA_i$ ($i \in I$) representing all coalgebras of F on objects of power less than λ (up to isomorphism of coalgebras). We have $|I| \leq \lambda$. Indeed, for every cardinal $n < \lambda$ let $I_n \subseteq I$ be the subset of all i with A_i having power n . Given $i \in I_n$, for every component $b: B \rightarrow A_i$ of A_i we know, since $\lambda > w(\mathcal{K})$, that there are at most λ morphisms from B to FA_i (recalling that FA_i has power at most λ), see (b) in Definition 4. Thus there are at most $n \cdot \lambda = \lambda$ morphisms from A_i to FA_i . And the number of objects A_i with n components is at most $w(\mathcal{K})^n < \lambda^n = \lambda$ (see Remark 12(d)). Thus, there are at most λ indexes in I_n . Since $I = \bigcup_{n < \lambda} I_n$, this proves $|I| \leq \lambda^2 = \lambda$.

Consequently $A = \coprod_{i \in I} A_i$ is an object of $\mathcal{K}_{\leq \lambda}$. We have the coalgebra structure $\alpha: A \rightarrow FA$ of a coproduct of (A_i, α_i) in $\mathbf{Coalg} F$. Let $e: A \rightarrow T$ be the wide pushout of all homomorphisms in $\mathbf{Coalg} F$ with domain (A, α) carried by epimorphisms of \mathcal{K} . Since \mathcal{K} is cocomplete and cowellpowered, and since the forgetful functor from $\mathbf{Coalg} F$ to \mathcal{K} creates colimits, this means that we form the corresponding pushout in \mathcal{K} and get a unique coalgebra structure $\tau: T \rightarrow FT$ making e a homomorphism:

$$\begin{array}{ccc} \coprod A_i & \xrightarrow{\alpha} & F(\coprod A_i) \\ e \downarrow & & \downarrow Fe \\ T & \xrightarrow{\tau} & FT \end{array}$$

The power of T is at most λ since T is a quotient of A , see (b) in Definition 4. We are going to prove that (T, τ) is a terminal coalgebra.

For every coalgebra $\beta: B \rightarrow FB$ with B having power less than λ there exists a unique homomorphism into (T, τ) . Indeed, the existence is clear: compose the isomorphism that exists from (B, β) to some (A_i, α_i) , the i -th coproduct injection of (A, α) and the above homomorphism e . To prove uniqueness, observe that by definition of (T, τ) , this coalgebra has no nontrivial quotient: every homomorphism with domain (T, τ) whose underlying morphism is epic in \mathcal{K} is invertible. Given homomorphisms $u, v: (B, \beta) \rightarrow (T, \tau)$

$$\begin{array}{ccc} B & \xrightarrow{\beta} & FB \\ u \downarrow & & \downarrow Fu \\ v \downarrow & & \downarrow Fv \\ T & \xrightarrow{\tau} & FT \\ q \downarrow & & \downarrow Fq \\ Q & \dashrightarrow & FQ \end{array}$$

form their coequalizer $q: T \rightarrow Q$ in \mathcal{K} . Then Q carries the structure of a coalgebra making q a homomorphism. Thus, q is invertible, proving $u = v$.

From Proposition 11 we deduce that the same holds for *all* coalgebras, thus (T, τ) is terminal. \blacktriangleleft

3 Algebraically Complete Categories

All the concrete categories proved to be algebraically cocomplete above turn out to be algebraically complete, too. Moreover, General Continuum Hypothesis need not be assumed for this result.

► **Remark 14.** In this remark we assume that, for a given ordinal λ , all (co)limits mentioned below exist. We denote by 0 the initial object and by 1 the terminal one.

- (a) Recall from [1] the *initial-algebra λ -chain* of an endofunctor F : its objects $F^i 0$ for all ordinals $i \leq \lambda + 1$ and its connecting morphisms $w_{ij}: F^i 0 \rightarrow F^j 0$ for all $i \leq j \leq \lambda + 1$ are defined by transfinite recursion as follows: $F^0 0 = 0$, $F^{i+1} 0 = F(F^i 0)$, and $F^j 0 = \operatorname{colim}_{i < j} F^i 0$ for limit ordinals $j \leq \lambda$. Analogously: $w_{01}: 0 \rightarrow F 0$ is unique, $w_{i+1, j+1} = F w_{ij}$, and w_{ij} ($i < j$) is a colimit cocone for every limit ordinal $j \leq \lambda$.
- (b) The initial-algebra chain *converges* at λ if the connecting map $w_{\lambda, \lambda+1}$ is invertible. In that case we get the initial-algebra

$$\mu F = F^\lambda 0$$

with the algebra structure $\iota = w_{\lambda, \lambda+1}^{-1}$

- (c) In particular, if F preserves colimits of λ -chains for a limit ordinal λ , then $\mu F = F^\lambda 0$.
- (d) Dually, the *terminal-coalgebra λ -chain* has objects $F^i 1$ (for $i \leq \lambda + 1$) with $F^0 1 = 1$, $F^{i+1} 1 = F(F^i 1)$ and $F^j 1 = \lim_{i < j} F^i 1$ for limit ordinals $j \leq \lambda$. Its connecting morphisms are denoted by v_{ij} ($i \geq j$). If F preserves limits of λ^{op} -chains, then $\nu F = F^\lambda 1$. This was explicitly formulated by Barr [8].
- (e) We say that a set functor F *preserves inclusion* if given a subset Y of X , then FY is a subset of FX , and for the inclusion map $i: Y \rightarrow X$ also $F i$ is the inclusion map. It follows that F preserves monomorphisms.

For every set functor F there exists a set functor G preserving inclusion and having the same initial-algebra chain as F for all infinite ordinals. Moreover, F and G coincide on all nonempty sets and functions and if $F\emptyset \neq \emptyset$, then $G\emptyset \neq \emptyset$. See [7, Theorem III.4.5] and [4, Remark 3]. We call G the *Trnková hull* of F .

► **Remark 15.** Let λ be an infinite regular cardinal. We recall from [6] that an object A of a category \mathcal{K} is called *λ -presentable* if its hom-functor $\mathcal{K}(A, -)$ preserves λ -filtered colimits. This means that if a λ -filtered diagram D has a colimit cocone $b_i: B_i \rightarrow X$ ($i \in I$), then for every morphism $a: A \rightarrow X$ (i) a factorization through b_i exists for some $i \in I$ and (ii) given two factorizations $u, v: A \rightarrow B_i$ with $a = b_i \cdot u = b_i \cdot v$, some connecting morphism $d: B_i \rightarrow B_j$ of D fulfils $d \cdot u = d \cdot v$.

A category \mathcal{K} is called *locally λ -presentable* if it is cocomplete and has a small full subcategory \mathcal{D} consisting of λ -presentable objects whose closure under λ -filtered colimits is all of \mathcal{K} . This implies that every object X is a canonical colimit of the diagram of all morphisms $a: A \rightarrow X$ with $A \in \mathcal{D}$. More precisely, of the λ -filtered diagram

$$D_X: \mathcal{D}/X \rightarrow \mathcal{D}, \quad D_X(A, a) = A.$$

In the case $\lambda = \aleph_0$ we speak about *locally finitely presentable* categories.

► **Definition 16** (See [5]). A *strictly locally λ -presentable* category is a locally λ -presentable category in which every morphism $b: B \rightarrow A$ with B λ -presentable has a factorization $b = b' \cdot f \cdot b$ for some morphisms $b': B' \rightarrow A$ and $f: A \rightarrow B'$ with B' also λ -presentable.

► **Examples 17** (See [5]).

- (a) The categories **Set**, **K -Vec** and **G -Set**, where G is a finite group, are strictly locally finitely presentable.
- (b) **Nom** is strictly locally \aleph_1 -presentable.
- (c) **Set^S** is strictly locally λ -presentable for infinite $\lambda > |S|$.
- (d) Given an infinite group G , the category **G -Set** is strictly locally λ -presentable if $\lambda > 2^{|G|}$.

► **Definition 18.** A category \mathcal{K} has *strict width* $w(\mathcal{K})$ if it has width $w(\mathcal{K})$, coproduct injections are monic, and every connected object is λ -presentable for $\lambda = w(\mathcal{K}) + \aleph_0$.

► **Example 19.**

- (1) The category **Set^S** has strict width $|S| + \aleph_0$, since connected objects (see Example 3) are finitely presentable.
- (2) **K -Vec** has strict width $|K| + \aleph_0$: the only connected object K is finitely presentable.
- (3) **G -Set** has strict width at most $2^{|G|} + \aleph_0$.
- (4) **Nom** has strict width \aleph_0 .

► **Lemma 20.** *If a category has strict width $w(\mathcal{K})$, then for every infinite regular cardinal $\lambda \geq w(\mathcal{K})$ its λ -presentable objects are precisely those of power less than λ .*

Proof. If X is λ -presentable and $X = \coprod_{i \in I} X_i$ with connected objects X_i , then in case $\text{card } I < \lambda$ we have nothing to prove. And if $\text{card } I \geq \lambda$, form the λ -filtered diagram of all coproducts $\coprod_{j \in J} X_j$ where J ranges over subsets of I with $\text{card } J < \lambda$. Since $\mathcal{K}(X, -)$ preserves this colimit, there exists a factorization of id_X through one of the colimit injections $v: \coprod_{j \in J} X_j \rightarrow \coprod_{i \in I} X_i$. Now v is monic (by the definition of strict width) and split epic, hence it is an isomorphism. Thus, $X \simeq \coprod_{j \in J} X_j$ has power at most $\text{card } J < \lambda$.

Conversely, if X has power less than λ , then it is λ -presentable because every coproduct of less than λ objects which are λ -presentable is λ -presentable. ◀

► **Remark 21.** In every locally λ -presentable category \mathcal{K} all hom-functors of λ -presentable objects collectively reflect λ -filtered colimits. That is, given a λ -filtered diagram D with objects D_i ($i \in I$), then a cocone $c_i: D_i \rightarrow C$ of D is a colimit iff for every λ -presentable object Y the following holds: (i) every morphism $f: Y \rightarrow C$ factorizes through some c_i and (ii) given two such factorizations $u, v: Y \rightarrow C$, $c_i \cdot u = c_i \cdot v$, there exists a connecting morphism $d: D_i \rightarrow D_j$ of D with $d \cdot u = d \cdot v$. This is proved for $\lambda = \aleph_0$ in [5, Lemma 2.7], the general case is completely analogous.

► **Theorem 22.** *Let \mathcal{K} be a strictly locally α -presentable category with a strict width. Then $\mathcal{K}_{\leq \lambda}$ is algebraically complete for every cardinal $\lambda \geq \max(\alpha, w(\mathcal{K}))$.*

Proof. Following Remark 14, it is sufficient to prove that $\mathcal{K}_{\leq \lambda}$ has colimits of i -chains for all limit ordinals $i \leq \lambda$, and every endofunctor of $\mathcal{K}_{\leq \lambda}$ preserves colimits of λ -chains.

- (1) $\mathcal{K}_{\leq \lambda}$ has for every limit ordinal $i \leq \lambda$ colimits of i -chains $(B_j)_{j < i}$. In fact, let X be the colimit of that chain in \mathcal{K} , then we verify that X has power at most λ . Indeed, each B_j is a coproduct of at most λ connected objects, thus, $\coprod_{j < i} B_j$ is a coproduct of at most $i \cdot \lambda = \lambda$ connected objects. The same holds for X , since it is a quotient of $\coprod_{j < i} B_j$.

- (2) For every endofunctor F of $\mathcal{K}_{\leq\lambda}$ and every λ -chain B_i ($i < \lambda$) in \mathcal{K}_{\leq} we prove that F preserves the colimit

$$X = \operatorname{colim}_{i \in I} B_i \quad (\text{with cocone } b_i: B_i \rightarrow X, i < \lambda).$$

Let us choose a small subcategory \mathcal{D} of \mathcal{K} as in Remark 15. We verify that the functor $B: \lambda \rightarrow \mathcal{D}/X$ given by $i \mapsto (B_i, b_i)$ is cofinal, i.e., for every object (A, a) of \mathcal{D}/X (a) there exists a morphism of \mathcal{D}/X into some (B_i, b_i) and (b) given a pair of morphisms $u, v: (A, a) \rightarrow (B_i, b_i)$, there exists $j \geq i$ with u and v merged by the connecting morphism $b_{ij}: B_i \rightarrow B_j$ of our chain. Indeed, since A is λ -presentable, the morphism $a: A \rightarrow \operatorname{colim}_{i < \lambda} B_i$ factorizes through b_i for some $i < \lambda$. And since u, v above fulfil $b_i \cdot u = b_i \cdot v (= a)$, some connecting morphism b_{ij} also merges u and v , see Remark 15.

Consequently, in order to prove that F preserves the colimit $X = \operatorname{colim} B_i$, it is sufficient to verify that it preserves the colimit of the codomain restriction $D'_X: \mathcal{D}/X \rightarrow \mathcal{K}_{\leq}$ of D_X (see Remark 15). Indeed, since $B: \lambda \rightarrow \mathcal{D}/X$ is cofinal, the colimits of the diagrams $F \cdot D'_X$ and $(FB_i)_{i < \lambda}$ coincide. We apply Remark 21 and verify the conditions (i) and (ii) for the cocone $Fa: FA \rightarrow FX$ of $F \cdot D_X$ (in \mathcal{K}). Thus $FX = \operatorname{colim} F \cdot D_X$ in \mathcal{K} which implies $FX = \operatorname{colim} FD'_X$ in $\mathcal{K}_{\leq\lambda}$.

Ad (i) Given a morphism $f: Y \rightarrow FX$ with Y λ -presentable, then Y has power less than λ , thus, by Proposition 10 there exists a coproduct injection $c: C \rightarrow X$ with C λ -presentable such that f factorizes through Fc (which is a member of our cocone).

Ad (ii) Let $u, v: Y \rightarrow FA$, with A λ -presentable, fulfil $Fa \cdot u = Fa \cdot v$. We are to find a connecting morphism

$$h: (A, a) \rightarrow (B, b) \quad \text{in } \mathcal{D}/X \quad \text{with } Fh \cdot u = Fh \cdot v.$$

By the strictness of \mathcal{K} , since A is λ -presentable, for $a: A \rightarrow X$ there exist morphisms $b: B \rightarrow X$ and $f: X \rightarrow B$ with B λ -presentable and $a = b \cdot f \cdot a$. It is sufficient to put $h = f \cdot a: A \rightarrow B$. Then h is a morphism of \mathcal{D}/X since $b \cdot h = a$, and $Fa \cdot u = Fb \cdot v$ implies $Fh \cdot u = Fh \cdot v$, as desired. \blacktriangleleft

► **Example 23.**

- (1) For every uncountable regular cardinal λ the category $\mathbf{Set}_{\leq\lambda}$ is algebraically complete (by Theorem 22) and, assuming GCH, algebraically cocomplete (by Theorem 13). The former was already proved in [3], Example 14, using an entirely different method.
- (2) The category $\mathbf{Set}_{\leq\aleph_0}$ of countable sets is algebraically complete, but not algebraically cocomplete. Indeed, the restriction \mathcal{P}_f of the finite power-set functor to it does not have a terminal coalgebra. Assuming that a (countable) terminal coalgebra T is given, we find a contradiction as follows. For every subset A of \mathbb{N} denote by C_A the tree with root r_A obtained from an infinite path by adding, for every number $n \in A$, a leaf of height $n + 1$. These trees are, as coalgebras for \mathcal{P}_f , clearly pairwise non-bisimilar. Consequently, the unique homomorphisms $h_A: C_A \rightarrow T$ have the property that the elements $h_A(r_A)$ are pairwise distinct. This is the desired contradiction: T is countable, but the number of all A 's is uncountable.

► **Example 24.** Let λ be an uncountable regular cardinal. The following categories are algebraically complete and, assuming GCH, algebraically cocomplete:

- (a) $\mathbf{Set}_{\leq\lambda}^S$ whenever $\lambda > |S|$,
 (b) $K\text{-Vec}_{\leq\lambda}$ whenever $\lambda > |K|$,

12:10 On Terminal Coalgebras Derived from Initial Algebras

(c) $\mathbf{Nom}_{\leq \lambda}$ whenever $\lambda > \aleph_1$, and

(d) $G\text{-Set}_{\leq \lambda}$ for groups G with $\lambda > 2^{|G|}$.

This follows from Theorems 13 and 22.

4 Terminal Coalgebras Derived from Initial Algebras

In this section we prove that a set functor F with a non-empty initial algebra of regular cardinality λ (see Remark 12) has a terminal coalgebra of the same cardinality λ – with one exception: $\lambda = \aleph_0$. We first formulate a fixed-point theorem.

A *fixed point* of an endofunctor F is an object X isomorphic to FX .

► **Theorem 25** (The Fixed-Point Theorem). *Assume GCH. A set functor with a nonempty fixed point of a finite or regular uncountable cardinality λ has a terminal coalgebra of cardinality at most λ .*

Proof.

(1) Without loss of generality we can assume $F\emptyset = \emptyset$. Indeed, otherwise we prove the theorem for the Trnková hull G , see Remark 14. The terminal coalgebras for F and G are the same.

F restricts to an endofunctor F_0 of $\mathbf{Set}_{\leq \lambda}$. Indeed, if A is a fixed point with $|A| = \lambda$, then every object $Y \neq \emptyset$ of $\mathbf{Set}_{\leq \lambda}$ is a split subobject of A , hence, FY is a split subobject of FA , proving that $|FY| \leq |FA| = \lambda$. We know from Theorem 13 that F_0 has a terminal coalgebra. We prove that this is also terminal for F . For that, it is sufficient to prove every coalgebra for F is a colimit of coalgebras for F_0 in $\mathbf{Coalg} F$.

(2) Suppose that λ is finite. Then we verify that the terminal coalgebra is obtained as the limit of the following ω^{op} -chain

$$1 \xleftarrow{!} F1 \xleftarrow{F!} F^2 1 \xleftarrow{F^2!} \dots$$

Indeed, since by (1) we have $|F^n 1| \leq \lambda$ for all n , there exists $k \leq \lambda$ such that some infinite set $A \subseteq \mathbb{N}$ fulfils $|F^n 1| = k$ for every $n \in A$. Observe that the connecting maps of our chain are all epic. Hence, given $n \geq m$ in A , the connecting map from $F^n 1$ to $F^m 1$ is invertible: it is monic due to $|F^n 1| = |F^m 1|$. Thus, the limit of the cofinal subchain $F^n 1$ ($n \in A$) is absolute, since this subchain consists of isomorphisms. Hence, the original chain also has an absolute limit. This implies by Remark 14(d), that $\lim_{n < \omega} F^n 1$ is a terminal coalgebra of F . It has $k \leq \lambda$ elements.

(3) From now on we assume that λ is uncountable. For every coalgebra $\alpha: A \rightarrow FA$ and every subset $b: B \hookrightarrow A$ with $|B| < \lambda$ a subset $b': B' \hookrightarrow A$ exists which contains b , fulfils $|B'| < \lambda$, and carries the structure $\beta': B' \rightarrow FB'$ of a subcoalgebra (i.e., $b': (B', \beta') \rightarrow (A, \alpha)$ is a coalgebra homomorphism). This is proved precisely as Proposition 11. It then follows that the diagram of all subcoalgebras of (A, α) on less than λ elements (and all coalgebra homomorphisms carried by inclusion maps) has the canonical λ -filtered colimit (A, α) in $\mathbf{Coalg} F$. Indeed, the forgetful functor U from $\mathbf{Coalg} F$ to \mathbf{Set} creates colimits, and A is (in \mathbf{Set}) a canonical λ -filtered colimit of all subsets of less than λ elements. The subdiagram of all subalgebras of less than λ elements is cofinal in the above diagram, hence, it also has the canonical colimit A . And U creates that colimit. ◀

► **Example 26.** None of the assumptions of the Fixed-Point Theorem can be left out, as we now demonstrate.

- (1) Assuming the negation of the Continuum Hypothesis, i.e. $\aleph_1 < 2^{\aleph_0}$, we present a set functor F with the fixed point \aleph_1 (the set of all countable ordinals) which has no terminal coalgebra. Define F on objects by

$$FX = X \times \aleph_1 + \{Y \subseteq X; |Y| > \aleph_1 \text{ or } Y = \emptyset\}.$$

For every morphism $f: X \rightarrow X'$ the left-hand summand of Ff is $f \times \text{id}_{\aleph_1}$, and the right-hand one is given by $Ff(\emptyset) = \emptyset$ and $Ff(Y) = f[Y]$ if f restricted to Y is monic, else \emptyset .

Then \aleph_1 is a fixed point of F : $F\aleph_1 = \aleph_1 \times \aleph_1 + \{\emptyset\} \cong \aleph_1$. But assuming that a terminal coalgebra $\tau: \nu F \rightarrow F(\nu F)$ exists, we derive a contradiction. It is clear that every fixed point of F has power \aleph_1 , thus $|\nu F| = \aleph_1$.

For every function $\varphi: \mathbb{N} \rightarrow \aleph_1$ define a coalgebra $A_\varphi = (\mathbb{N}, \alpha_\varphi)$ for F as follows: α_φ maps n to the element $(n + 1, \varphi(n))$ of the left-hand summand of FX . We have a unique coalgebra homomorphism $h_\varphi: A_\varphi \rightarrow \nu F$. Since $\tau \cdot h_\varphi = Fh_\varphi \cdot \alpha_\varphi$, for every $n \in \mathbb{N}$ we get

$$\tau(h_\varphi(n)) = Fh_\varphi(n + 1, \varphi(n)) = (h_\varphi(n + 1), \varphi(n)). \tag{4.1}$$

We conclude that the elements $h_\varphi(0) \in \nu F$ for all $\varphi: \mathbb{N} \rightarrow \aleph_1$ are pairwise distinct: assuming $h_\varphi(0) = h_{\varphi'}(0)$, we prove $\varphi = \varphi'$. Indeed, it is sufficient to verify that $h_\varphi(n) = h_{\varphi'}(n)$ by induction on n . This is trivial, the induction hypothesis yields, due to (4.1), $(h_\varphi(n + 1), \varphi(n)) = (h_{\varphi'}(n + 1), \varphi'(n))$, and the left-hand components prove $h_\varphi(n + 1) = h_{\varphi'}(n + 1)$.

This is a desired contradiction: all elements $h_\varphi(0) \in T$ form a set of power at most $|T| = \aleph_1$, but all $\varphi: \mathbb{N} \rightarrow \aleph_1$ form a set of power $|\aleph_1^{\mathbb{N}}| \geq 2^{\aleph_0} > \aleph_1$.

- (2) For the non-regular uncountable cardinal $\aleph_\omega = \bigvee_{n < \omega} \aleph_n$ we present a set functor F with the fixed point \aleph_ω not having a terminal coalgebra. Since by Remark 12(b) we have $|\aleph_\omega^{\mathbb{N}}| > \aleph_\omega$, this is completely analogous to the preceding example: put $FX = X \times \aleph_\omega + \{Y \subseteq X; |Y| > \aleph_\omega \text{ or } Y = \emptyset\}$.
- (3) The Fixed-Point Theorem does not hold for \aleph_0 , see Example 1.

► **Theorem 27 (The Terminal-Coalgebra Theorem).** *Assume GCH. If a set functor F has a nonempty initial algebra of a finite or regular uncountable cardinality, then it also has a terminal coalgebra of the same cardinality. Shortly:*

$$\mu F \simeq \nu F.$$

Proof. In [7], Theorem 3.10, it is proved that the existence of μF implies that the initial-algebra chain $F^i 0$ ($i \in \text{Ord}$), see Remark 14, converges, thus $\mu F = F^\rho 0$ for some ordinal ρ . Without loss of generality we assume $\rho \geq \omega$. Put $\lambda = |\mu F|$.

By the Fixed-Point Theorem we have a terminal coalgebra $\tau: T \rightarrow FT$ with $|T| \leq \lambda$. The algebra (T, τ^{-1}) yields, as proved in [1], a unique cocone $\alpha_i: F^i 0 \rightarrow T$ of the initial-algebra chain satisfying $\alpha_{i+1} = \tau^{-1} \cdot F\alpha_i$ for every ordinal i . To prove $|T| \geq \lambda$, we verify that α_i is monic for all ordinals $i \geq \omega$. Thus $|T| \geq |F^i 0|$ for all i , proving $|T| \geq |F^\rho 0| = \lambda$.

Let F preserve monomorphisms. Then all α_i ($i \in \text{Ord}$) are monic. This is easily seen by transfinite induction, since $\alpha_0: \emptyset \rightarrow A$ is monic, and $\alpha_{i+1} = \tau^{-1} \cdot F\alpha_i$ is monic whenever α_i is.

12:12 On Terminal Coalgebras Derived from Initial Algebras

For a functor F not preserving monomorphisms we have $F\emptyset \neq \emptyset$, since all nonempty monomorphisms split. We apply the result of Remark 14(e) that the Trnková hull is a monic-preserving set functor G which coincides with F on all nonempty sets (and functions) and whose initial-algebra chain is, from the ordinal ω onwards, the same as that for F . Thus, $\mu G \simeq \mu F$. We also have $\nu G \simeq \nu F$ since, due to $F\emptyset \neq \emptyset \neq G\emptyset$, F and G have the same coalgebras. ◀

► Example 28.

- (a) For set functors with countable initial algebras nothing can be deduced about the terminal coalgebras. As we have seen in Example 1, νF need not exist. And for every infinite cardinal λ there exists a set functor with a terminal coalgebra such that $|\mu F| = \aleph_0$ and $|\nu F| > \lambda$. Define F as the following subfunctor of the functor of Example 1:

$$FX = \mathcal{P}_f X + \{M \subseteq X; \aleph_0 < |M| \leq \lambda\}.$$

This functor has a terminal coalgebra because it preserves colimits of λ^+ -chains (see Remark 14). And νF is uncountable. Indeed, P_f has an uncountable terminal coalgebra: the argument is as in Example 23(2). Since \mathcal{P}_f is a subfunctor of F , the terminal-coalgebra chain of P_f is also a subfunctor of the terminal-coalgebra chain of F , from which we conclude $|\nu P_f| \leq |\nu F|$.

Furthermore, $\mu F \cong \mu \mathcal{P}_f$ is countable. And for every uncountable fixed point $X \simeq FX$ we clearly have $X \simeq \{M \subseteq X; |M| = \lambda\}$, therefore $|X| = |X|^\lambda$ from which it follows, by Remark 12, that $|X| > \lambda$. Hence, $|\nu F| > \lambda$.

- (b) For many-sorted sets the Terminal-Coalgebra Theorem does not hold. Indeed, given any cardinal λ there is an endofunctor F of $\mathbf{Set} \times \mathbf{Set}$ that fulfils: μF exists and has λ elements, but νF does not exist. Put

$$F(X, Y) = \begin{cases} (\emptyset, \lambda) & \text{if } X = \emptyset, \\ (X, \lambda + \mathcal{P}Y) & \text{else.} \end{cases}$$

Given a morphism $(f, g): (X, Y) \rightarrow (X', Y')$ with X nonempty, put $F(f, g) = (f, id + Pg)$. It is easy to see that the initial algebra of F is (\emptyset, λ) . If F would have a terminal coalgebra $\nu F = (A, B)$, then $A = \emptyset$ (since otherwise (A, B) is not a fixed point of F). But for any coalgebra $\alpha: (X, Y) \rightarrow (X, \lambda + \mathcal{P}Y)$, with $X \neq \emptyset$, no morphism into (\emptyset, B) exists, a contradiction.

- (c) Moreover, for every pair $\lambda_\mu \leq \lambda_\nu$ of infinite cardinals there exists an endofunctor F of $\mathbf{Set} \times \mathbf{Set}$ with μF of λ_μ elements and νF of λ_ν elements. On objects put $F(X, Y) = (\emptyset, \lambda_\mu)$ if $X = \emptyset$, else $(1, \lambda_\mu)$. To every morphism F assigns the (obvious) inclusion map.

Both the initial-algebra chain and the terminal-coalgebra chain converge in one step and yield $\mu F = (\emptyset, \lambda_\mu)$ and $\nu F = (1, \lambda_\nu)$.

5 Finitary Set Functors

In the preceding section we have established, for some set functors F , an isomorphism $\mu F \simeq \nu F$. But that concerned only the underlying sets! In the generality of that section, nothing can be derived about the relationship of the algebra structure $\iota: F(\mu F) \rightarrow \mu F$ and the coalgebra structure $\tau: \nu F \rightarrow F(\nu F)$. For finitary set functors F (i.e., those preserving filtered colimits) with $F\emptyset \neq \emptyset$ we can say more. Firstly, μF and νF exist, and μF (considered as a coalgebra via ι^{-1}) is a subcoalgebra of νF . Second, there is a canonical ultrametric on νF , such that for the metric subspace μF we prove that

- (a) μF and νF share the same Cauchy completion,
and
- (b) ι determines τ as the unique continuous extension of ι^{-1} .

This generalizes the result of Barr [8] that, in case F moreover preserves limits of ω^{op} -chains, νF is the Cauchy completion of μF .

► **Proposition 29** (μF as a subcoalgebra of νF). *If a set functor F has a terminal coalgebra, then it also has an initial algebra carried by a subset $\mu F \subseteq \nu F$ such that the inclusion map $m: \mu F \hookrightarrow \nu F$ is the unique coalgebra homomorphism, i.e., $\tau \cdot m = Fm \cdot \iota^{-1}$.*

Proof.

- (1) Assume first that F preserves monomorphisms. There exists a unique cocone of the initial-algebra chain with codomain νF , $m_i: F^i 0 \rightarrow \nu F$ ($i \in \text{Ord}$) determined by the condition below:

$$m_{i+1} \equiv F(F^i 0) \xrightarrow{Fm_i} F(\nu F) \xrightarrow{\tau^{-1}} \nu F \quad (i \in \text{Ord}).$$

Easy transfinite induction verifies that m_i is monic for every i . Since νF has only a set of subobjects, there exists an ordinal λ such that all m_i with $i \geq \lambda$ represent the same subobject. Thus the commutative triangle below

$$\begin{array}{ccc} W_\lambda & \xrightarrow{w_{\lambda, \lambda+1}} & FW_\lambda \\ & \searrow m_\lambda & \swarrow m_{\lambda+1} \\ & \nu F & \end{array}$$

implies that $w_{\lambda, \lambda+1}$ is invertible. Consequently, the following algebra

$$F(F^\lambda 0) \xrightarrow{w_{\lambda, \lambda+1}^{-1}} F^\lambda 0$$

is initial, see Remark 14.

For the monomorphism $m_\lambda: F^\lambda 0 \rightarrow \nu F$ put

$$\mu F = m_\lambda[F^\lambda 0] \subseteq \nu F.$$

Choose an isomorphism $r: \mu F \rightarrow F^\lambda 0$ such that $m = m_\lambda \cdot r: \mu F \rightarrow \nu F$ is the inclusion map. Then there exists a unique algebra structure $\iota: F(\mu F) \rightarrow \mu F$ for which r is an isomorphism of algebras:

$$r: (\mu F, \iota) \xrightarrow{\sim} (F^\lambda 0, w_{\lambda, \lambda+1}^{-1}).$$

The following commutative diagram

$$\begin{array}{ccc} \mu F & \xrightarrow{\iota^{-1}} & F(\mu F) \\ r \downarrow & & \downarrow Fr \\ F^\lambda 0 & \xrightarrow{w_{\lambda, \lambda+1}} & F(F^\lambda 0) \\ m_\lambda \downarrow & \nearrow m_{\lambda+1} & \downarrow Fm_\lambda \\ \nu F & \xrightarrow{\tau} & F(\nu F) \end{array}$$

proves that $m = m_\lambda \cdot r$ is the unique coalgebra homomorphism, as required.

12:14 On Terminal Coalgebras Derived from Initial Algebras

- (2) Let F not preserve monomorphisms. Therefore $F\emptyset \neq \emptyset$. Our proposition holds for the Trnková hull G of Remark 14(e). Since F and G agree on all nonempty sets and $F\emptyset \neq \emptyset \neq G\emptyset$, they have the same terminal coalgebras. Since the initial algebra of G is, as we have just seen, obtained via the initial-algebra chain, and F has from ω onwards the same initial-algebra chain, F and G have the same initial algebras. Thus, our proposition holds for F too. \blacktriangleleft

The fact that every set functor with a terminal coalgebra has an initial algebra was proved in [15]. Our proof above uses ideas of that paper.

Next we recall the behaviour of the terminal-coalgebra chain, see Remark 14, for finitary set functors:

► **Theorem 30** (Worrell [16]). *For every finitary set functor F the terminal-coalgebra chain converges at $\omega + \omega$: $\nu F = F^{\omega+\omega}1$. Moreover, every connecting morphism $v_{i,\omega}$ ($i \geq \omega$) is monic.*

► **Remark 31.**

- (a) Consequently, νF is a canonical subset of $F^\omega 1 = \lim_{n < \omega} F^n 1$. And this endows νF with a canonical ultrametric, as our next lemma explains. Recall that a metric d is called an *ultrametric* if for all elements x, y, z the triangle inequality can be strengthened to $d(x, z) \leq \max(d(x, y), d(y, z))$.
- (b) For every set functor F there exists a unique morphism $\bar{u}: F^\omega 0 \rightarrow F^\omega 1$ with $\bar{u} \cdot w_{n,\omega} = v_{\omega,n} \cdot F^n 1$ (where $!: 0 \rightarrow 1$ is unique). See [2], Lemma 2.4.
- (c) The homomorphism m of Proposition 29 fulfils $\bar{u} = v_{\omega+\omega,\omega} \cdot m$. Indeed, since F is finitary, we have $\mu F = F^\omega 0$ and $\iota = w_{\omega,\omega+1}^{-1}$. Thus m being a coalgebra homomorphism states precisely that

$$v_{\omega+\omega+1,\omega+1}^{-1} \cdot m = Fm \cdot w_{\omega,\omega+1}$$

or, $m = v_{\omega+\omega+1,\omega+\omega} \cdot Fm \cdot w_{\omega,\omega+1}$. The squares defining \bar{u} in Remark 31(b) thus commute when \bar{u} is substituted by $v_{\omega+\omega,\omega} \cdot m$ ($= v_{\omega+\omega+1,\omega} \cdot Fm \cdot w_{\omega,\omega+1}$). That is, we claim that

$$v_{\omega,n} [v_{\omega+\omega+1,\omega} \cdot Fm \cdot w_{\omega,\omega+1}] \cdot w_{n,\omega} = F^n !$$

This is clear for $n = 0$. If this holds for n , i.e., if

$$v_{\omega+\omega+1,n} \cdot Fm \cdot w_{n,\omega+1} = F^n !,$$

then it also holds for $n + 1$: just apply F to that equation. Thus, $\bar{u} = v_{\omega+\omega} \cdot m$.

► **Lemma 32.** *Every limit L of an ω^{op} -chain in **Set** carries a complete ultrametric: assign to $t \neq s$ in L the distance 2^{-n} where n is the least natural number with $p_n(t) \neq p_n(s)$ for the limit projections p_n .*

Proof. Let $l_n: L \rightarrow A_n$ ($n \in \mathbb{N}$) be a limit cone. For the above function

$$d(x, y) = 2^{-n}$$

where $l_n(x) \neq l_n(y)$ and n is the least such number we see that d is symmetric. It satisfies the ultrametric inequality

$$d(x, z) \leq \max(d(x, y), d(y, z)) \quad \text{for all } x, y, z \in L.$$

This is obvious if the three elements are not pairwise distinct. If they are, the inequality follows from the fact that if l_n separates two elements, then so do all l_m with $m \geq n$.

It remains to prove that the space $F^\omega 1$ is complete. Given a Cauchy sequence $x_r \in L$ ($r \in \mathbb{N}$), for every $k \in \mathbb{N}$ there exists $r(k) \in \mathbb{N}$ with

$$d(x_{r(k)}, x_n) < 2^{-k} \quad \text{for every } n \geq r(k).$$

Choose $r(k)$'s to form an increasing sequence. Then $d(x_{r(k)}, x_{r(k+1)}) < 2^{-k}$, i.e., $l_k(x_{r(k)}) = l_k(x_{r(k+1)})$. Therefore, the elements $y_k = l_k(x_{r(k)})$ are compatible: we have $a_{k+1}(y_{k+1}) = y_k$ for all $k \in \mathbb{N}$. Consequently, there exists a unique $y \in L$ with $l_k(y) = y_k$ for all $k \in \mathbb{N}$. That is, $d(y, x_{r(k)}) < 2^{-k}$. Thus, y is the desired limit:

$$y = \lim_{k \rightarrow \infty} x_{r(k)} \quad \text{implies} \quad y = \lim_{n \rightarrow \infty} x_n. \quad \blacktriangleleft$$

We conclude that for a finitary set functor both νF and μF carry a canonical ultrametric: νF as a subspace of $F^\omega 1$ via $v_{\omega+\omega, \omega} : \nu F \rightarrow F^\omega 1$, and μF as a subspace of νF via m . Or, equivalently, a subspace of $F^\omega 1$ via \bar{u} , see Remark 31. Given $t \neq s$ in νF we have $d(t, s) = 2^{-n}$ for the least $n \in \mathbb{N}$ with $v_{\omega+\omega, n}(t) \neq v_{\omega+\omega, n}(s)$.

► **Notation 33.** Given a finitary set functor F with $F\emptyset \neq \emptyset$, choose an element $p : 1 \rightarrow F\emptyset$. This defines the following morphisms for every $n \in \mathbb{N}$:

$$e_n = \bar{u} \cdot w_{n+1, \omega} \cdot F^n p : F^n 1 \rightarrow F^\omega 1.$$

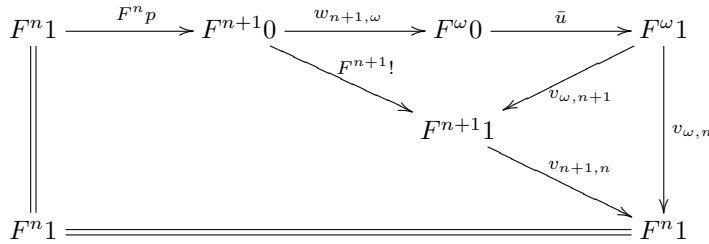
We also put $r_n = e_n \cdot v_{\omega, n} : F^\omega 1 \rightarrow F^n 1$.

► **Observation 34.** Denote by $! : \emptyset \rightarrow 1$ the unique map. For every $n \in \mathbb{N}$ we have

(a) $v_{n, n+1} \cdot F^{n+1} ! \cdot F^n p = id_{F^n 1}$.

This is obvious for $n = 0$. The induction step just applies F to the given square.

(b) $v_{\omega, n} \cdot e_n = id_{F^n 1}$. Indeed, in the following diagram



the upper right-hand part commutes by the definition of \bar{u} , see Remark 31(b), the left-hand one does by (a), and the lower right-hand triangle is clear.

(c) $v_{\omega, n} \cdot r_n = v_{\omega, n}$. This follows from (b): precompose it with $v_{\omega, n}$.

► **Theorem 35.** For a finitary set functor F with $F\emptyset \neq \emptyset$ the Cauchy completions of the ultrametric spaces μF and νF coincide. And the algebra structure ι determines the coalgebra structure τ as the unique continuous extension of ι^{-1} .

Proof.

(1) Assume first that F preserves inclusion, see Remark 14(e).

(a) We prove that the subset $\bar{u} = v_{\omega+\omega, \omega} \cdot m : \mu F \rightarrow F^\omega 1$ of Remark 31(b) is dense in $F^\omega 1$, thus, the complete space $F^\omega 1$ is a Cauchy completion of both $\bar{u}[\mu F]$ and $v_{\omega+\omega, \omega}[\nu F]$.

12:16 On Terminal Coalgebras Derived from Initial Algebras

For every $x \in F^\omega 1$ the sequence $r_n(x)$ lies in the image of $e_n \cdot v_{\omega,n}$ which, in view of the definition of e_n , is a subset of the image of \bar{u} . And we have $x = \lim_{n \rightarrow \infty} r_n(x)$ because Observation 34 (c) yields $v_{\omega,n}(x) = v_{\omega,n}(r_n(x))$. Thus $d(x, r_n(x)) < 2^{-n}$ for all $n \in \mathbb{N}$.

(b) We have ultrametric subspaces μF and νF of $F^\omega 1$, hence, the bijections

$$F(\mu F) \xrightarrow{\iota} \mu F \quad \text{and} \quad \nu F \xrightarrow{\tau} F(\nu F)$$

make also $F(\mu F)$ and $F(\nu F)$ ultrametric spaces. The continuous map ι^{-1} has at most one continuous extension to νF , since μF is dense in νF (even in $F^\omega 1$). And τ is such an extension: it is not only continuous, it is an isometry. And it extends ι^{-1} by Proposition 29: choose an inclusion map m with $\tau \cdot m = Fm \cdot \iota^{-1}$. Since Fm is an inclusion map, τ is an extension of ι^{-1} .

(2) Once we have established (a) and (b) for inclusion-preserving finitary functors, it holds for all finitary functors F . Indeed, use the Trnková hull G that agrees with F on all nonempty set and functions with $G\emptyset \neq \emptyset$ provided that $F\emptyset \neq \emptyset$, see Remark 14(e). Consequently, the coalgebras for F and G coincide. And the initial-algebra chains coincide for infinite ordinals, in particular $F^\omega 0 = G^\omega 0$, that is, F and G have the same initial algebra. \blacktriangleleft

► Example 36.

(1) For the set functor $FX = X \times \Sigma + 1$ (of dynamic systems with inputs from Σ and deadlock states) the terminal coalgebra is obtained in ω steps, since F preserves limits of ω^{op} -sequences. It can be described as the coalgebra $\nu F = \Sigma^\infty$ of all finite and infinite words over Σ . The distance of distinct words u and v is 2^{-n} for the largest n such that u and v have the same prefix of length n .

The initial algebra Σ^* is dense in Σ^∞ : every infinite word is the limit of the sequence of its finite prefixes. The algebra structure $\iota: \Sigma^* \times \Sigma + 1 \rightarrow \Sigma^*$ is given by concatenation on the left-hand summand, and the empty word on the right-hand one. Its inverse has a unique continuous extension to Σ^∞ assigning to every nonempty word u the pair $(\text{head}(u), \text{tail}(u))$. This is indeed the coalgebra structure of νF .

(2) For the finite power-set functor \mathcal{P}_f the initial algebra can be described as $\mu \mathcal{P}_f =$ all finite extensional trees (where trees are considered up to isomorphism), see [16]. Recall that a tree is called *extensional* if for every node x the maximum subtrees of x are pairwise non-isomorphic. And it is called *strongly extensional* if it has no nontrivial tree bisimulation; for finite trees these two concepts are equivalent. Worrell proved in [16] that the terminal coalgebra $\nu \mathcal{P}_f$ consists of all finitely branching strongly extensional trees, whereas \mathcal{P}_f^ω consists of all strongly extensional trees. The metric on $\mathcal{P}_f^\omega 1$ assigns to trees $t \neq s$ the distance $d(t, s) = 2^{-n}$, where n is the least number with $\partial_n t \neq \partial_n s$. Here $\partial_n t$ is the extensional tree obtained from t by cutting it at level n and forming the extensional quotient of the resulting tree.

The algebraic structure $\iota: \mathcal{P}_f(\mu \mathcal{P}_f) \rightarrow \mathcal{P}_f$ assigns to a set $\{t_1, \dots, t_n\}$ of finite trees the tree-tupling (consisting of a new root and n maximum subtrees t_1, \dots, t_n). The coalgebraic structure $\tau: \nu \mathcal{P}_f \rightarrow \mathcal{P}_f(\nu \mathcal{P}_f)$ assigns to a tree $t \in \nu \mathcal{P}_f$ the finite set of its maximum subtrees. This is indeed a continuous extension of ι^{-1} .

6 Conclusions and Open problems

Whereas a set functor is known to have an initial algebra iff it has a fixed point, for terminal coalgebras fixed points are not sufficient in general. However, we have proved that a non-empty fixed point of a finite or regular cardinality λ implies that a terminal coalgebra exists

and has at most λ elements – with a single exception, $\lambda = \aleph_0$. From this fixed-point result we have derived that every set functor F with a nonempty initial algebra μF whose cardinality is finite or regular uncountable has a terminal coalgebra $\nu F \cong \mu F$.

We have also presented a number of categories that are algebraically complete and cocomplete, i.e., every endofunctor has a terminal coalgebra and an initial algebra. Examples include (for sufficiently large regular cardinals λ) the category $\mathbf{Set}_{\leq \lambda}$ of sets of power at most λ , $\mathbf{Nom}_{\leq \lambda}$ of nominal sets of power at most λ , $K\text{-Vec}_{\leq \lambda}$ of vector spaces of dimension at most λ , and $G\text{-Set}_{\leq \lambda}$ of G -sets (where G is a group) of power at most λ .

All these results assumed the General Continuum Hypothesis. It is an open question what could be proved without this assumption. Another question is whether the above relationship $\nu F \cong \mu F$ can, under suitable side conditions, be proved for more general base categories than \mathbf{Set} .

For finitary set functors F with $F\emptyset \neq \emptyset$ we have presented a sharper result: both μF and νF carry a canonical ultrametric and these two spaces have the same Cauchy completion. Moreover, by inverting the algebra structure of μF we obtain the coalgebra structure of νF as the unique continuous extension.

References

- 1 J. Adámek. Free algebras and automata realizations in the language of categories. *Comment. Math. Univ. Carolinae*, 15:589–602, 1974.
- 2 J. Adámek. Final coalgebras are ideal completions of initial algebras. *J. Logic Comput.*, 12:217–242, 2002.
- 3 J. Adámek and V. Koubek. Least fixed point of a functor. *J. Comput. System Sciences*, 19:163–168, 1979.
- 4 J. Adámek and V. Koubek. On the greatest fixed point of a set functor. *Theoret. Comput. Sci.*, 150:57–75, 1995.
- 5 J. Adámek, S. Milius, L. Sousa, and T. Wissmann. On finitary functors and finitely presentable algebras. *CoRR*, 2019. arXiv:1902.05788.
- 6 J. Adámek and J. Rosický. *Locally presentable and accessible categories*. Cambridge University Press, 1994.
- 7 J. Adámek and V. Trnková. *Automata and Algebras in Categories*. Kluwer Acad. Publ., London, 1990.
- 8 M. Barr. Terminal coalgebras in well-founded set theory. *Theoret. Comput. Sci.*, 114:299–315, 1993.
- 9 P. Freyd. Algebraically complete categories. *Lecture Notes in Math.*, 1488:95–104, 1970.
- 10 T. Jech. *Set theory*. Academic Press, 1978.
- 11 J. Rutten. Universal coalgebra. *Theoret. Comput. Sci.*, 249:3–80, 2000.
- 12 M.B. Smyth and G.D. Plotkin. The category-theoretic solution of recursive domain equations. *SIAM J. Comput.*, 11:761–788, 1982.
- 13 A. Tarski. Sur la décomposition des ensembles en sous-ensembles presque disjoint. *Fund. Math.*, 14:205–215, 1929.
- 14 V. Trnková. On descriptive classification of set-functors I., II. *Comment. Math. Univ. Carolinae*, 12:143–175 and 345–357, 1971.
- 15 V. Trnková, J. Adámek, V. Koubek, and J. Reiterman. Free algebras, input processes and free monads. *Comment. Math. Univ. Carolinae*, 15:589–602, 1974.
- 16 J. Worrell. On the final sequence of a finitary set functor. *Theoret. Comput. Sci.*, 338:184–199, 2005.

A Full proofs

PROOF OF EXAMPLES 5 (3) and (4).

- (1) For 5(4) recall that objects of **G-Set** are pairs (X, \cdot) where X is a set and \cdot is a function from $G \times X$ to X such that

$$h(gx) = (hg)x \quad \text{for } h, g \in G \text{ and } x \in X,$$

and

$$ex = x \quad \text{for } x \in X \text{ (} e \text{ neutral in } G \text{)}.$$

An important example is given by any equivalence relation \sim on G which is *equivariant*, i.e., fulfils

$$g \sim g' \Rightarrow hg \sim hg' \quad \text{for all } g, g', h \in G.$$

Then the quotient set G/\sim is a G -set (of equivalence classes $[g]$) w.r.t. the action $g[h] = [gh]$. This G -set is clearly connected.

- (2) Let (X, \cdot) be a G -set. For every element $x \in X$ we obtain a subobject of (X, \cdot) on the set

$$Gx = \{gx; g \in G\} \quad \text{(the orbit of } x \text{)}.$$

The equivalence on G given by

$$g \sim g' \quad \text{iff } gx = g'x$$

is equivariant, and the G -sets Gx and G/\sim are isomorphic. Moreover, two orbits are disjoint or equal: given $gx = hy$, then $x = (g^{-1}h)y$, thus, $Gx = Gy$.

- (3) Every object (X, \cdot) is a coproduct of at most $|X|$ connected objects: if X_0 is a choice class of the equivalence $x \equiv y$ iff $Gx = Gy$, then

$$X = \coprod_{x \in X_0} Gx.$$

- (4) The number of connected objects, up to isomorphism, is at most $2^{|G|} + \aleph_0$. Indeed, it follows from the above that the connected objects are represented by precisely all G/\sim where \sim is an equivariant equivalence relation. If $|G| = \beta$ then we have at most β^β equivalence relations. For β infinite, this is equal to $2^{|\beta|}$, for β finite, this is smaller than $2^{|\beta|} + \aleph_0$.

- (5) The number of morphisms from G/\sim to an object (X, \cdot) is at most $|X| \leq \max(\alpha, 2^{|\beta|} + \aleph_0)$ where $\alpha = |X_0|$ in (3) above. Indeed, every morphism p is determined by the value $x_0 = p([e])$ since $p([g]) = p(g[e]) = g \cdot x_0$ holds for all $[g] \in G/\sim$.

- (6) Finally, for 5(3) the proof is completely analogous: in (2) each orbit $S_f(\mathbb{A})/\sim \simeq S_f(\mathbb{A})x$ is a nominal set. And the number of all such orbits up to isomorphism is \aleph_0 , see Lemma A1 in [5]. In (5) we have $|X| \leq \alpha \cdot \aleph_0 = \alpha$ for all $\alpha \geq \aleph_0$. ◀

PROOF OF PROPOSITION 10. It is sufficient to prove this in case X has power precisely λ (otherwise put $c = \text{id}_X$). And we can assume that B is connected. In the general case we have $B = \coprod_{k \in K} B_k$ with $|K| < \lambda$, and find for each k a summand $c_k: C_k \rightarrow X$ corresponding to the k -th component of b . Then we let $c: C \rightarrow X$ be the least summand containing each c_k . (C has power less than λ since each C_k does and $|K| < \lambda$.)

Since $\lambda > w(K)$, in the coproduct of λ connected objects representing X at least one, say R , must appear λ times. Thus X has the form

$$X = \coprod_{\lambda} R + X_0$$

for objects R and X_0 , with R connected. Let \bar{X}_0 be the coproduct of the same components as in X_0 , but each taken precisely once. Thus

- (a) \bar{X}_0 has power at most $w(\mathcal{K})$, and
- (b) we have a coproduct injection

$$m: \bar{X}_0 \rightarrow X_0$$

which has an (obvious) splitting

$$\hat{m}: X_0 \rightarrow \bar{X}_0, \quad \hat{m} \cdot m = \text{id}.$$

Put

$$Y = \coprod_{\lambda} R + \bar{X}_0$$

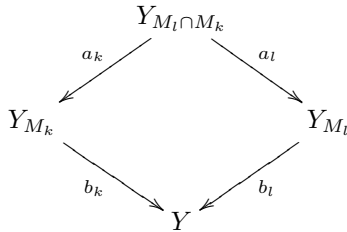
and for every set $M \subseteq \lambda$ put

$$Y_M = \coprod_M R + \bar{X}_0.$$

(c) By Remark 8(b) we can choose $t \in \lambda$ and an almost disjoint collection of sets $M_k \subseteq \lambda$, $k \in K$, with

$$t \in M_k, \quad |M_k| = \lambda \quad \text{and} \quad |K| > \lambda.$$

Consider the following square of coproduct injections for any pair $k, l \in K$:



This is an absolute pullback. Indeed, it obviously commutes. And b_k and b_l are split monomorphisms: define

$$\bar{b}_k: Y \rightarrow Y_{M_k}$$

as identity on the summand \bar{X}_0 , whereas the i -th copy of R is sent to copy i , if $i \in M_k$, and to copy t else. Then

$$\bar{b}_k b_k = \text{id}.$$

Analogously for b_l . Next define

$$\bar{a}_l: Y_{M_l} \rightarrow Y_{M_k \cap M_l}$$

12:20 On Terminal Coalgebras Derived from Initial Algebras

as identity on the summand \bar{X}_0 , whereas the i -th copy of R is sent to copy i , if $i \in M_k$, and to copy t else. Then clearly

$$\bar{a}_k a_l = \text{id} \quad \text{and} \quad a_k \bar{a}_l = \bar{b}_k b_l.$$

Thus, the above square is an absolute pullback by Lemma 6.

(d) We are ready to prove that for a connected object B every morphism

$$b: B \rightarrow FX$$

has the required factorization. For every $k \in K$ since $|M_k| = \lambda$ we have an isomorphism

$$y_k: Y = \coprod_{\lambda} R + \bar{X}_0 \longrightarrow \coprod_{M_k} R + \bar{X}_0 = Y_{M_k}$$

which composed with $b_k: Y_{M_k} \rightarrow Y_{\lambda}$ yields an endomorphism

$$z_k = b_k \cdot y_k: Y \rightarrow Y.$$

We use (b) above and precompose z_k with $\tilde{m} = \text{id} + \hat{m}: X \rightarrow Y$ to get the following morphisms

$$B \xrightarrow{b} FX \xrightarrow{F\tilde{m}} FY \xrightarrow{Fz_k} FY \quad (k \in K).$$

They are not pairwise distinct because $|K| > \lambda$, whereas FY has at most λ components (since F is an endofunctor of $\mathcal{K}_{\leq \lambda}$) so that (b) in Definition 4 implies that $\mathcal{K}(B, FY)$ has cardinality at most λ . Choose $k \neq l$ in K with

$$Fz_k \cdot F\tilde{m} \cdot b = Fz_l \cdot F\tilde{m} \cdot b. \tag{A.1}$$

Compare the pullbacks Z of z_k and z_l and $Y_{M_k \cap M_l}$ of b_k and b_l :

$$\begin{array}{ccccc}
 & & Z & & \\
 & & \downarrow p & & \\
 & p_k \swarrow & & \searrow p_l & \\
 Y & & Y_{M_k \cap M_l} & & Y \\
 \downarrow y_k & \swarrow a_k & & \searrow a_l & \downarrow y_l \\
 Y_{M_k} & & & & Y_{M_l} \\
 & \searrow b_k & & \swarrow b_l & \\
 & & Y_{\lambda} & &
 \end{array}$$

Since y_k and y_l are isomorphisms, the connecting morphism p between the above pullbacks is an isomorphism, too. We know that $|M_k \cap M_l| < \lambda$ since M_k, M_l are of our almost disjoint family, thus the object

$$C = Y_{M_k \cap M_l} = \coprod_{M_k \cap M_l} R + \bar{X}_0$$

has less than λ summands, as required. And, due to (c), the pullback of z_k and z_l is absolute. The equality (A.1) thus implies that $F\tilde{m} \cdot b$ factorizes through Fp_k :

$$\begin{array}{ccccc}
 & & B & & \\
 & & \downarrow b & & \\
 & & FX & & \\
 & \nearrow h & \uparrow F\tilde{m} & \downarrow F[\text{id}+m] & \\
 FZ & \xrightarrow{Fp_k} & FY & \xrightarrow[\cong]{Fz_k} & FY \\
 & & \downarrow Fz_l & &
 \end{array}$$

Consequently, from $\hat{m}m = \text{id}$ we obtain

$$b = F\tilde{m} \cdot Fp_k \cdot h = F\tilde{m} \cdot Fp_k \cdot Fp^{-1} \cdot Fp \cdot h.$$

Thus, for the coproduct injection

$$c \equiv \tilde{m} \cdot p_k \cdot p^{-1}: C \rightarrow X,$$

we get the desired factorization $b = Fc \cdot (Fp \cdot h)$. ◀

Coinductive Resumption Monads: Guarded Iterative and Guarded Elgot

Paul Blain Levy 

University of Birmingham, UK
P.B.Levy@cs.bham.ac.uk

Sergey Goncharov 

FAU Erlangen-Nürnberg, Germany
Sergey.Goncharov@fau.de

Abstract

We introduce a new notion of “guarded Elgot monad”, that is a monad equipped with a form of iteration. It requires every guarded morphism to have a specified fixpoint, and classical equational laws of iteration to be satisfied. This notion includes Elgot monads, but also further examples of partial non-unique iteration, emerging in the semantics of processes under infinite trace equivalence.

We recall the construction of the “coinductive resumption monad” from a monad and endofunctor, that is used for modelling programs up to bisimilarity. We characterize this construction via a universal property: if the given monad is guarded Elgot, then the coinductive resumption monad is the guarded Elgot monad that freely extends it by the given endofunctor.

2012 ACM Subject Classification Theory of computation → Categorical semantics; Theory of computation → Axiomatic semantics

Keywords and phrases Guarded iteration, guarded monads, coalgebraic resumptions

Digital Object Identifier 10.4230/LIPIcs.CALCO.2019.13

Funding *Sergey Goncharov*: Support by Deutsche Forschungsgemeinschaft (DFG) under project GO 2161/1-2 is gratefully acknowledged.

1 Introduction

The study of monads for effects has developed in numerous directions since it was initiated in [18]. We make two contributions to this research area. Firstly we give a new notion of “guarded Elgot monad” – a monad equipped with a form of iteration – that includes a variety of examples. Secondly, we give a universal property for one of these examples, the so-called “coinductive resumption monad”. We shall explain these contributions separately.

1.1 Monads and Iteration

Monads. Let us recall the basic ideas of monads for effects, where the base category is **Set**. A monad **T** on **Set**, presented in “Kleisli triple” form, consists of three things.

- For each set X , a set TX , of which an element represents a “computation” that may perform various computational effects and may return an element of X .
- For each set X , a map $\eta_X: X \rightarrow TX$. For $x \in X$, the image $\eta_X(x)$ represents a “pure computation” that just returns x .
- For any map $f: X \rightarrow TY$, we have a map $f^*: TX \rightarrow TY$. For $p \in TX$, the image $f^*(p)$ represents a “sequenced computation” that first executes p and then, if this returns $x \in X$, proceeds to execute $f(x) \in TY$.

These must satisfy three equations, as described in [18]. A map $X \rightarrow TY$ is called a *Kleisli map*, and these form the *Kleisli category*, denoted $\text{Kl}(\mathbf{T})$. It inherits coproducts from **Set**.



© Paul Blain Levy and Sergey Goncharov;

licensed under Creative Commons License CC-BY

8th Conference on Algebra and Coalgebra in Computer Science (CALCO 2019).

Editors: Markus Roggenbach and Ana Sokolova; Article No. 13; pp. 13:1–13:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

13:2 Coinductive Resumption Monads: Guarded Iterative and Guarded Elgot

Monads for printing. We give (in outline) some example monads for computations that print characters. Let A be an alphabet, i.e. a set of characters. We write A^* (resp. A^ω , $A^{\leq\omega}$) for the set of finite sequences (resp. infinite sequences, finite and infinite sequences). Here are our examples.

- The monad $X \mapsto A^* \times X$ represents computations that print several characters and then return a value.
- The monad $X \mapsto A^* \times X + A^{\leq\omega}$ represents such computations, but also computations that continue forever and never return. The latter includes computations that print finitely many characters and then diverge (i.e. hang), and also computations that print infinitely many characters.
- The monad $X \mapsto A^* \times X + A^\omega$ represents computations that may return or continue forever, but in the latter case are required to be “productive”, i.e. keep printing.

For our next series of examples, write \mathcal{P}^+X for the set of nonempty subsets of X . The following are monads for *nondeterministic* printing computations.

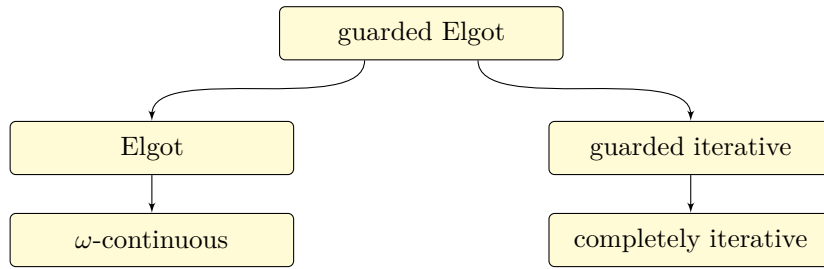
$$\begin{aligned} X &\mapsto \mathcal{P}^+(A^* \times X) \\ X &\mapsto \mathcal{P}^+(A^* \times X + A^{\leq\omega}) \\ X &\mapsto \mathcal{P}^+(A^* \times X + A^\omega) \end{aligned}$$

A nondeterministic printing computation has *terminating traces* in $A^* \times X$, *divergences* in A^* and *infinite traces* in A^ω . (A similar arrangement has been used in CSP semantics [21].) The above monads identify computations that are *infinite trace equivalent*, i.e. that have the same terminating traces, divergences and infinite traces.

Iterative computations. Given a Kleisli map $f: X \rightarrow Y + X$, we would like to form a Kleisli map $f^\dagger: X \rightarrow Y$ where, for $x \in X$, the image $f^\dagger(x)$ represents the following “iterative computation”. First it executes $f(x) \in T(Y + X)$. That may return $\text{inl } y$, in which case the iterative computation returns y , or it may return $\text{inr } x'$, in which case the computation represented by $f(x') \in T(Y + X)$ is executed, and so forth. Can we form f^\dagger for our example monads?

- For the monad $X \mapsto A^* \times X + A^{\leq\omega}$, we can form f^\dagger , since the monad is able to represent infinite computations.
- For the monad $X \mapsto \mathcal{P}^+(A^* \times X + A^{\leq\omega})$, f^\dagger is formed analogously.
- For the monad $X \mapsto A^* \times X + A^\omega$, we can form f^\dagger , provided f is *guarded*. That means that for all $x \in X$, the image $f(x) \in A^* \times (X + Y) + A^\omega$ is not of the form $\text{inl}(\varepsilon, \text{inl } x')$ for some $x' \in X$. This condition ensures that $f^\dagger(x)$ represents a productive computation, because an iterative call is possible only after at least one character has been printed.
- For the monad $\mathcal{P}^+(A^* \times X + A^\omega)$, f^\dagger is formed analogously. Here the guardedness requirement is that, for all $x \in X$, the image $f(x) \in \mathcal{P}^+(A^* \times (X + Y) + A^\omega)$ does not contain $\text{inl}(\varepsilon, \text{inr } x')$ for any $x' \in X$.

These four examples motivate the first contribution of the paper, viz. the notion of a *guarded Elgot monad*. This consists of a monad on a co-Cartesian category \mathcal{C} (i.e. category with finite coproducts), equipped with two additional structures. Firstly a *guardedness predicate*, that tells us when a Kleisli map $f: X \rightarrow Y + Z$ is guarded in the right summand. When this condition holds, we write $f: X \rightarrow Y \rangle Z$. Secondly, a *guarded Conway operator* that associates to each map $f: X \rightarrow Y \rangle X$ a Kleisli map $f^\dagger: X \rightarrow Y$. Each of these structures



■ **Figure 1** Connections between classes of monads with iteration.

must satisfy some conditions that we shall stipulate. In particular, for $f: X \rightarrow Y \rangle X$, we require f^\dagger to be a *fixpoint* of f , i.e. a Kleisli map $g: X \rightarrow Y$ such that

$$\begin{array}{ccc}
 X & \xrightarrow{f} & Y + X \\
 & \searrow g & \downarrow [\text{id}, g] \\
 & & Y
 \end{array}$$

commutes in the Kleisli category.

Although the above four examples are all guarded Elgot monads, they are significantly different.

- The guarded Elgot monads $X \mapsto A^* \times X + A^{\leq \omega}$ and $X \mapsto \mathcal{P}^+(A^* \times X + A^{\leq \omega})$ are special because *every* Kleisli map $f: X \rightarrow Y + Z$ is deemed to be guarded in the right summand. So, for every Kleisli map $f: X \rightarrow Y + X$, we can form f^\dagger . We call these simply *Elgot monads*. (They are called “complete Elgot monads” in [7].)
- The guarded Elgot monad $X \mapsto A^* \times X + A^\omega$ is special because, for each map $f: X \rightarrow Y \rangle X$, the map f^\dagger is the *unique* fixpoint of f . We call this a *guarded iterative monad* [10].
- For $A \neq \perp$, the guarded Elgot monad $X \mapsto \mathcal{P}^+(A^* \times X + A^\omega)$ is neither Elgot nor guarded iterative. (This is proved in Example 20(5) below). So it illustrates the need for the new, more general notion of guarded Elgot monad.

As noted in [10], *every* monad can be regarded as guarded iterative, by saying that a Kleisli map $f: X \rightarrow Y + Z$ is “vacuously” guarded in the right summand when it factorizes via $\text{inr}: Z \rightarrow Y + Z$.

1.2 Resumption Monads

Let us write $\mu\gamma.F\gamma$ for an initial algebra of F , and $\nu\gamma.F\gamma$ for a final coalgebra. We note the following.

- The set $A^* \times X$ can be written $\mu\gamma.(X + H\gamma)$, where H is the endofunctor $Y \mapsto A \times Y$.
- The set $A^* \times X + A^\omega$ can be written $\nu\gamma.(X + H\gamma)$.
- The set $A^* \times X + A^{\leq \omega}$ can be written $\nu\gamma.\text{Maybe}(X + H\gamma)$, where $\text{Maybe } Y \stackrel{\text{def}}{=} Y + 1$.

More generally, given a monad T and endofunctor H on a co-Cartesian category \mathcal{C} , we form two monads:

- the *inductive resumption monad* \mathbf{T}_H^μ sending $X \mapsto \mu\gamma.T(X + H\gamma)$, provided these initial algebras exist [6]
- the *coinductive resumption monad* \mathbf{T}_H^ν sending $X \mapsto \nu\gamma.T(X + H\gamma)$, provided these final coalgebras exist [20].

13:4 Coinductive Resumption Monads: Guarded Iterative and Guarded Elgot

For example, with $\mathcal{C} = \mathbf{Set}$, let \mathbf{T} be the countable nonempty powerset monad and let $H: Y \mapsto A \times Y$. Then these monads represent countably nondeterministic printing computations modulo bisimilarity. Here, the difference between \mathbf{T}_H^μ and \mathbf{T}_H^ν is that the former represents only computations that eventually return a value, whereas the latter represents also computations that continue forever (but are productive).

For another class of examples, let $(B(a))_{a \in A}$ be a “signature”, i.e. family of sets, and let H be the endofunctor $Y \mapsto \sum_{a \in A} Y^{B(a)}$. Again let \mathbf{T} be the countable nonempty powerset monad. In this case the monads \mathbf{T}_H^μ and \mathbf{T}_H^ν represent countably nondeterministic computations that perform I/O. Such a computation can print an element $a \in A$ and then pause; if the user then enters an element of $B(a)$, the computation resumes. This is the reason for the name “resumption monad”. The printing example is the special case where $B(a)$ is singleton for all $a \in A$.

As the above examples illustrate, these monads provide a natural way of combining an endofunctor (representing I/O) with a monad (representing other effects, e.g. nondeterminism). So one may ask of each monad: can it be characterized via a universal property?

This has been done for \mathbf{T}_H^μ in [15]. We recall this result, but present it a little differently, using the notion of *free extension* (defined in full generality in Definition 2 below).

1.3 Free Extensions

To explain the notion of free extension, we give a well-known example: the polynomial ring $R[X_0, X_1]$. This is the free extension of the ring R by the set $\{0, 1\}$. That means that we have a function and ring homomorphism

$$\{0, 1\} \xrightarrow{X_-} R[X_0, X_1] \longleftarrow R$$

(here X_- reads as a map sending $i \in \{0, 1\}$ to X_i) that are universal: for any function and ring homomorphism

$$\{0, 1\} \xrightarrow{g} S \longleftarrow h R$$

there is a unique mediating homomorphism

$$\begin{array}{ccc} \{0, 1\} & \xrightarrow{X_-} & R[X_0, X_1] & \longleftarrow & R \\ & \searrow g & \downarrow \text{dotted} & \swarrow h & \\ & & S & & \end{array}$$

We can now describe the result of [14] as follows: the monad \mathbf{T}_H^μ is a free extension of \mathbf{T} by H . This means that we have a natural transformation and monad morphism

$$H \xrightarrow{\beta} \mathbf{T}_H^\mu \longleftarrow \rho \mathbf{T}$$

that are universal.

The second contribution of this paper is the following analogous result. If \mathbf{T} is a guarded Elgot monad, then \mathbf{T}_H^ν is also guarded Elgot, and moreover it is the free extension, among guarded Elgot monads, of \mathbf{T} by H . This means that – in a suitable sense we shall define – we have a guarded natural transformation and guarded Elgot monad morphism

$$H \xrightarrow{\beta} \mathbf{T}_H^\nu \longleftarrow \rho \mathbf{T}$$

that are universal. This in turn gives a universal property for the two special cases simply by varying the notion of guardedness in which our result is parametric.

- If \mathbf{T} is Elgot, then \mathbf{T}_H^ν is Elgot, and therefore it is the free extension, among Elgot monads, of \mathbf{T} by H . This result appeared (with a considerably more complex proof) in [9].
- If \mathbf{T} is guarded iterative (as noted above, *any* monad can be so regarded), then \mathbf{T}_H^ν is guarded iterative [10], and therefore it is the free extension, among guarded iterative monads, of \mathbf{T} by H . A similar result – using “two-sided ideals” rather than guardedness predicates – was given in [19, Corollary 4.6], generalizing [16].

In general, a free extension of an initial object is a free object. (This is Proposition 4 below.) For example, the ring \mathbb{Z} of integers is initial among all rings, so $\mathbb{Z}[X_0, X_1]$ is a free ring on the set $\{0, 1\}$. This gives some more special cases.

- The identity monad is initial among all monads. So Id_H^μ is a free monad on H .
- The identity monad is initial among all guarded iterative monads, and among all guarded Elgot monads. So Id_H^ν is a free guarded iterative monad, and a free guarded Elgot monad, on H . With $H = \text{Id}$ this yields Capretta’s *delay monad* $\nu\gamma. - + \gamma$ used for modeling partiality in intensional type theory [5].
- On \mathbf{Set} , the Maybe monad is initial among all Elgot monads. This is true, more generally, on any *hyperextensive category* [1]. So Maybe_H^ν is a free completely Elgot monad on H . This was previously shown in [9].

It is also worth noting that free extensions can also be described as coproducts with free objects. (This is Proposition 5 below). For example, the free extension of a ring R by the set $\{0, 1\}$ can be described as the coproduct of R and the free ring on $\{0, 1\}$. This formulation is used in [14, 13, 19, 9] and indeed we provide a coproduct characterization in this style in Corollary 29 below. We take the view, however, the characterization in terms of free extensions is more primitive, since it does not require the free object to exist.

2 Preliminaries

In this paper we work in co-Cartesian categories, which are categories with finite coproducts. We fix selected coproduct co-spans $X \xrightarrow{\text{inl}} X + Y \xleftarrow{\text{inr}} Y$ and initial objects 0 with $[\]: 0 \rightarrow X$ denoting the initial morphisms. We do not generally assume *extensiveness*, in particular, the injections inl and inr need not be monic.

In a category \mathcal{C} , we denote by $|\mathcal{C}|$ the associated class of objects and by $\mathcal{C}(X, Y)$ the set of morphisms from X to Y . We occasionally omit indexes at natural transformation components to improve readability. For a functor $F: \mathcal{C} \rightarrow \mathcal{C}$, we denote by $(\nu F, \text{out}: \nu F \rightarrow F\nu F)$ the *final F -coalgebra*. Whenever possible, we use bold letters, e.g. \mathbf{T} , for monads, to emphasize the distinction with the underlying functor T . A monad \mathbf{T} over \mathcal{C} induces a *Kleisli category* $\text{Kl}(\mathbf{T})$ with $|\text{Kl}(\mathbf{T})| = |\mathcal{C}|$ and $\text{Kl}(\mathbf{T})(X, Y) = \mathcal{C}(X, TY)$. We make free use of the well-known fact that for a co-Cartesian \mathcal{C} and a monad \mathbf{T} on \mathcal{C} , the Kleisli category $\text{Kl}(\mathbf{T})$ is again co-Cartesian with the coproduct co-spans $X \xrightarrow{\eta \text{inl}} T(X + Y) \xleftarrow{\eta \text{inr}} Y$ and $[(T \text{inl})f, (T \text{inr})g]: X + Y \rightarrow T(X + Y)$ being the coproduct of morphisms $f: X \rightarrow TX'$ and $g: Y \rightarrow TY'$.

Unless stated otherwise, all diagrams we present are supposed to commute.

3 Free Extensions

We recall the following standard notion, see e.g. [3, Section 7.7].

► **Definition 1** (Bimodules). *For categories \mathcal{C} and \mathcal{D} , a bimodule $\mathcal{O}: \mathcal{C} \leftrightarrow \mathcal{D}$ consists of the following data:*

- a family of sets $(\mathcal{O}(X, \underline{Y}))_{X \in |\mathcal{C}|, \underline{Y} \in |\mathcal{D}|}$, where $g \in \mathcal{O}(X, \underline{Y})$ is called an \mathcal{O} -morphism $g: X \rightarrow \underline{Y}$;

13:6 Coinductive Resumption Monads: Guarded Iterative and Guarded Elgot

■ each $g: X \rightarrow \underline{Y}$ can be composed with a \mathcal{C} -map $f: X' \rightarrow X$ or \mathcal{D} -map $h: \underline{Y} \rightarrow \underline{Y}'$.
 For $g: X \rightarrow \underline{Y}$, $f': X'' \rightarrow X'$, $f: X' \rightarrow X$, $h': \underline{Y}' \rightarrow \underline{Y}''$, $h: \underline{Y} \rightarrow \underline{Y}'$ we must have the following:

$$\begin{aligned} g \text{id}_X &= g & (h' h)g &= h'(h g) & h(g f) &= (h g) f \\ \text{id}_{\underline{Y}} g &= g & g(f f') &= (g f) f' \end{aligned}$$

For example: the bimodule $\mathbf{Set} \rightarrow \mathbf{Ring}$ in which $\mathcal{O}(X, \underline{Y})$ is the set of functions from the set X to the ring \underline{Y} . This bimodule can be seen as arising from the forgetful functor $\mathbf{Ring} \rightarrow \mathbf{Set}$.

Bimodules $\mathcal{C} \rightarrow \mathcal{D}$ correspond to functors $\mathcal{C}^{\text{op}} \times \mathcal{D} \rightarrow \mathbf{Set}$. They are also called *distributors* or *profunctors* (but some authors reverse the direction). For the rest of the section, let $\mathcal{O}: \mathcal{C} \rightarrow \mathcal{D}$ be a bimodule.

► **Definition 2** (Free Extensions). Let $A \in |\mathcal{C}|$ and $\underline{B} \in |\mathcal{D}|$. A free extension of \underline{B} by A consists of $\underline{V} \in |\mathcal{D}|$ and $e: A \rightarrow \underline{V}$ and $f: \underline{B} \rightarrow \underline{V}$, such that, for all $\underline{X} \in |\mathcal{D}|$ and $g: A \rightarrow \underline{X}$ and $h: \underline{B} \rightarrow \underline{X}$, there is a unique $k: \underline{V} \rightarrow \underline{X}$ such that

$$\begin{array}{ccccc} A & \xrightarrow{e} & \underline{V} & \xleftarrow{f} & \underline{B} \\ & \searrow g & \vdots k & \swarrow h & \\ & & \underline{X} & & \end{array}$$

► **Definition 3** (Free Objects). Let $A \in |\mathcal{C}|$. A free object on A consists of $\underline{V} \in |\mathcal{D}|$ and $e: A \rightarrow \underline{V}$, such that, for all $\underline{X} \in |\mathcal{D}|$ and $g: A \rightarrow \underline{X}$, there is a unique $k: \underline{V} \rightarrow \underline{X}$ such that

$$\begin{array}{ccc} A & \xrightarrow{e} & \underline{V} \\ & \searrow g & \vdots k \\ & & \underline{X} \end{array}$$

► **Proposition 4.** Let $0_{\mathcal{D}}$ be an initial object in \mathcal{D} . For any $A \in |\mathcal{C}|$, a free object on A corresponds to a free extension of $0_{\mathcal{D}}$ by A . The bijection sends (\underline{V}, e) to

$$A \xrightarrow{e} \underline{V} \xleftarrow{[]} 0_{\mathcal{D}}.$$

► **Proposition 5.** Let $A \in |\mathcal{C}|$ and $\underline{B} \in |\mathcal{D}|$. Let (\underline{W}, d) be a free object on A . Then a coproduct of \underline{W} and \underline{B} corresponds to a free extension of \underline{B} by A . The bijection sends (\underline{V}, e, f) to

$$A \xrightarrow{d} \underline{W} \xrightarrow{e} \underline{V} \xleftarrow{f} \underline{B}.$$

4 Guardedness on Monads

In this section, let \mathcal{K} be a co-Cartesian category. The typical example is $\mathcal{K} = \mathbf{Kl}(\mathbf{T})$, where \mathbf{T} is a monad on a co-Cartesian category \mathcal{C} .

4.1 Guardedness Predicates

The following notion is slightly adapted from [10].

► **Definition 6** (Guardedness, Guarded Monads). A guardedness predicate on \mathcal{K} provides for all objects X, Y, Z a subset $\mathcal{K}^\bullet(X, Y, Z) \subseteq \mathcal{K}(X, Y + Z)$. We write $f: X \rightarrow Y \rangle Z$ for $f \in \mathcal{K}^\bullet(X, Y, Z)$ and say that f is guarded (in the right summand). The following conditions are required:

$$\begin{array}{l} \text{(trv)} \quad \frac{f: X \rightarrow Y}{\text{in}f: X \rightarrow Y \rangle Z} \qquad \text{(par)} \quad \frac{f: X \rightarrow V \rangle W \quad g: Y \rightarrow V \rangle W}{[f, g]: X + Y \rightarrow V \rangle W} \\ \\ \text{(cmp)} \quad \frac{f: X \rightarrow Y \rangle Z \quad g: Y \rightarrow V \rangle W \quad h: Z \rightarrow V + W}{[g, h]f: X \rightarrow V \rangle W} \end{array}$$

A category equipped with a guardedness predicate is called guarded category. A monad \mathbf{T} on \mathcal{C} is a guarded monad if $\mathcal{K} = \text{Kl}(\mathbf{T})$ is a guarded category under the coproducts inherited from \mathcal{C} .

We write “let $f: X \rightarrow Y \rangle Z$ ” as an abbreviation for “let f be a map $X \rightarrow Y + Z$ be a map such that $f: X \rightarrow Y \rangle Z$ ”.

Intuitively, a morphism $f: X \rightarrow Y \rangle Z$ represents a program flow with inputs in X and outputs in Y and in Z , where the latter part of the output is guarded in the sense that every portion of the program flow from X to Z runs through a guard. The notion of guard here is implicit and depends on the specific model. The axioms of guardedness abstractly capture properties of guards: **(trv)** states that if all the output goes to Y then $f: X \rightarrow Y + Z$ is (vacuously) guarded in Z ; **(par)** states that guardedness jointly depends on all inputs; finally, **(cmp)** states that if the program flow branches then every branch leading to the guarded output must hit a guard at least once, specifically, $h: Z \rightarrow V + W$ need not be guarded in W , because h receives the input from f , which ensures guarded already.

The distinction between Definition 6 and the corresponding definition in [10] is precisely determined by the choice of the notion of coproduct: in op. cit. coproducts are treated up to isomorphisms, while here we work with selected coproducts. The original axiomatization of guardedness additionally involved a *weakening rule*, which turned out to be derivable from the above three [8]. Let us summarize this and other consequences of the axioms. We will need the following convention.

► **Notation 7.** Let us use the notation $f: X \rightarrow Y \rangle Y_1 \rangle \dots \rangle Y_n$, for $f: X \rightarrow (\dots(Y + Y_1) + \dots) + Y_n$ meaning that $\sigma f: X \rightarrow Y \rangle Y_1 + \dots + Y_n$ where σ is the obvious associativity isomorphism $(\dots(Y + Y_1) + \dots) + Y_n \rightarrow Y + (Y_1 + (\dots + Y_n) \dots)$.

► **Proposition 8.** Let \mathcal{K} be a guarded category.

1. For all objects $V, W \in |\mathcal{K}|$, we have $[\]: 0 \rightarrow V \rangle W$.
2. Let $f: X \rightarrow Y \rangle Z$. For $u: X' \rightarrow X$ and $g: Y \rightarrow Y'$ and $h: Z \rightarrow Z'$ we have $(g + h)fu: X' \rightarrow Y' \rangle Z'$.
3. (Weakening) If $f: X \rightarrow Y \rangle Z \rangle W$ then $f: X \rightarrow Y + Z \rangle W$.

It is often useful to speak of guardedness in particular summands:

- we say that $f: X \rightarrow Y + Z$ is *inr-guarded* if $f: X \rightarrow Y \rangle Z$;
- we say that $f: X \rightarrow Y + Z$ is *inl-guarded* if $X \xrightarrow{f} Y + Z \cong Z + Y$ is *inr-guarded*;
- we say that $f: X \rightarrow Y$ is *id-guarded* if $X \xrightarrow{f} Y \cong 0 + Y$ is *inr-guarded*.

Two guardedness predicates are especially important.

► **Proposition 9** (Greatest and Least Guardedness Predicates).

1. The greatest guardedness predicate on \mathcal{K} says that, for every map $f: X \rightarrow Y + Z$, we have $f: X \rightarrow Y \rangle Z$.
2. The least guardedness predicate on \mathcal{K} says that, for $f: X \rightarrow Y + Z$, $f: X \rightarrow Y \rangle Z$ iff there is a map $g: X \rightarrow Y$ such that f factors as $X \xrightarrow{g} Y \xrightarrow{\text{inl}} Y + Z$ (such g need not be unique, since it does not follow from our running premises that coproduct injections are monic).

We say that \mathcal{K} is *totally guarded* when equipped with the largest guardedness predicate, and *vacuously guarded* when equipped with the smallest.

► **Example 10.** Here are some examples of guardedness predicates for $\mathcal{K} = \text{Kl}(\mathbf{T})$ with \mathbf{T} being a monad on \mathbf{Set} .

1. Let \mathbf{T} be the following monad: $T\emptyset = \emptyset$ and $TX = 1$ if $X \neq \emptyset$, under vacuous guardedness. Now, the unique morphism $1 \rightarrow T(1 + 1) = 1$ is inl-guarded and inr-guarded, because it factors through $1 = T1 \xrightarrow{T \text{ inr}} T(1 + 1) = 1$ and through $1 = T1 \xrightarrow{T \text{ inl}} T(1 + 1) = 1$. But $1 \rightarrow T(1 + 1) = 1$ does not factor through $\emptyset = T\emptyset \xrightarrow{T[\cdot]} T(1 + 1) = 1$, and hence it is not id-guarded. This example show that guardedness in two summands does not necessarily imply guardedness in their union.
2. Let \mathcal{P}^+ be the non-empty powerset monad. For $f: X \rightarrow \mathcal{P}^+(Y + Z)$, say $f: X \rightarrow Y \rangle Z$ when for every $x \in X$, the set $f(x)$ contains at least one element of the form inl y .
3. Let \mathcal{D}^+ be the *countable probability distribution monad*:

$$\mathcal{D}^+ X = \left\{ d : X \rightarrow [0, 1] \mid \sum d = 1 \right\}.$$

We put $f: X \rightarrow Y \rangle Z$ if for every $x \in X$, $f(x)(\text{inl } y) > 0$ for at least one $y \in Y$.

4. For a set A , let $TX = A^* \times X$ be a *writer monad* whose monad structure is induced by the monoid structure of A^* . For $f: X \rightarrow A^* \times (Y + Z)$, say $f: X \rightarrow Y \rangle Z$ when, for every $x \in X$, if $f(x) = (m, \text{inr } z)$ then $m \neq \varepsilon$.
5. Following Section 1.1, let A be again an arbitrary set and let $TX = \mathcal{P}^+(A^* \times X + A^\omega)$. This yields a monad for nondeterministic programs that print characters in A , giving semantics that records the (successful) finite and infinite traces. The monad structure is obtained from the fact that A^* is a monoid and A^ω is a left A^* -module. For $f: X \rightarrow \mathcal{P}^+(A^* \times (Y + Z) + A^\omega)$, say $f: X \rightarrow Y \rangle Z$ when, for every $x \in X$, if $(m, \text{inr } z) \in f(x)$ then $m \neq \varepsilon$. Intuitively, as in the previous example, a program denoting f is prohibited from returning a value through Z without first printing a character.

► **Definition 11** (Guarded Natural Transformations and Monad Morphisms).

1. Let \mathbf{T} and \mathbf{S} be guarded monads on \mathcal{C} . A monad morphism $\rho: \mathbf{T} \rightarrow \mathbf{S}$ is guarded when the functor $\text{Kl}(\rho): \text{Kl}(\mathbf{T}) \rightarrow \text{Kl}(\mathbf{S})$ preserves guardedness. Explicitly: for $f: X \rightarrow T(Y + Z)$, if $f: X \rightarrow Y \rangle Z$ in $\text{Kl}(\mathbf{T})$ then $X \xrightarrow{f} T(Y + Z) \xrightarrow{\rho_{Y+Z}} S(Y + Z)$ is guarded $X \rightarrow Y \rangle Z$ in $\text{Kl}(\mathbf{S})$.
2. Let H be an endofunctor and \mathbf{T} a guarded monad on \mathcal{C} . A natural transformation $\sigma: H \rightarrow T$ is guarded when for all $X \in |\mathcal{C}|$, $\sigma_X: HX \rightarrow TX$ is id-guarded.

4.2 Guarded Iteration

We now consider when guarded morphisms can be iterated in the sense of Section 1.1. The most straightforward case is the following:

► **Definition 12** (Guarded Iterative Categories). \mathcal{K} is guarded iterative if every $f: X \rightarrow Y \rangle X$ has a unique fixpoint $f^\dagger: X \rightarrow Y$ of the map $[\text{id}, -] f: \mathcal{K}(X, Y) \rightarrow \mathcal{K}(X, Y)$.

► **Lemma 13.** In any guarded category, if $f: X \rightarrow Y \rangle Z \rangle X$ and $g: X \rightarrow Y$ is a fixpoint of $[\text{id}, -] f$ then $g: X \rightarrow Y \rangle Z$.

► **Definition 14** (Conway Iteration). A guarded Conway (iteration) operator on \mathcal{K} associates to each $f: X \rightarrow Y \rangle X$ a fixpoint $f^\dagger: X \rightarrow Y$ of the map $[\text{id}, -] f$, satisfying the following principles:

- naturality: for $f: X \rightarrow Y \rangle X$ and $g: Y \rightarrow Z$ we have $((g + \text{id})f)^\dagger = gf^\dagger$;
- dinaturality: $([\text{inl}, h]g)^\dagger = [\text{id}, ([\text{inl}, g]h)^\dagger]g$ for $g: X \rightarrow Y \rangle Z$ and $h: Z \rightarrow Y \rangle X$ or $g: X \rightarrow Y + Z$ and $h: Z \rightarrow Y \rangle X$;
- codiagonal: $([\text{id}, \text{inr}]f)^\dagger = f^{\dagger\dagger}$ for $f: X \rightarrow Y \rangle X \rangle X$.

Note that in the codiagonal equation, $f^{\dagger\dagger}$ must exist by Lemma 13.

► **Remark 15.** Guarded Conway operators are direct generalizations of standard (total) Conway operators [2, 22], which arise under the total notion of guardedness. It was observed by Hyland and Hasegawa [12, 11] that Conway operators are equivalent to monoidal trace operators under $\otimes = +$ (modulo the duality of $+$ and \times). The connection between Conway operators and traces extends to a connection between guarded Conway operators and *guarded traces* [8]. In the total case, it is known that the axioms of Conway operators are incomplete wrt nontrivial models of iteration, e.g. the category of pointed complete partial orders [22]. This led Bloom and Ésik to completing the axiomatization of iteration by an infinite set of axioms called *commutative identities* [2]. These identities are instance of a single versatile quasi-equational *uniformity* principle, which holds true in all non-pathological models.

Let $J: \mathcal{C} \rightarrow \mathcal{K}$ be a functor, where \mathcal{C} and \mathcal{K} are guarded and have the same objects, and J is identity-on-objects and strictly preserves co-Cartesian structure.

► **Definition 16** (Uniformity). A guarded Conway operator $-^\dagger$ on \mathcal{K} is uniform (wrt J) when for \mathcal{K} -maps $f: X \rightarrow Y \rangle X$ and $g: Z \rightarrow Y \rangle Z$ and \mathcal{C} -map $h: Z \rightarrow X$,

$$\begin{array}{ccc} Z & \xrightarrow{g} & Y + Z \\ Jh \downarrow & & \downarrow Y+Jh \\ X & \xrightarrow{f} & Y + X \end{array} \quad \Rightarrow \quad \begin{array}{ccc} Z & \xrightarrow{g^\dagger} & Y \\ Jh \downarrow & \nearrow f^\dagger & \\ X & & \end{array}$$

► **Proposition 17.** [10] An operation sending every $f \in \mathcal{K}^\bullet(X, Y, Z)$ to a fixpoint $f^\dagger \in \mathcal{K}(X, Y)$ is guarded Conway uniform iff it satisfies naturality, codiagonal and uniformity. In other words, dinaturality is derivable.

► **Proposition 18.** Let \mathcal{K} be guarded iterative. Then $f \mapsto f^\dagger$ is a guarded Conway operator and uniform wrt $\text{Id}_{\mathcal{K}}$.

Proof. Except for uniformity wrt $\text{Id}_{\mathcal{K}}$, the proof is in [10, Theorem 17]. Let us verify the missing case of uniformity. Suppose that $f(Jh) = J(\text{id} + h)g$ for suitable f, g and h . Now, $[\text{id}, f^\dagger(Jh)]g = [\text{id}, f^\dagger]J(\text{id} + h)g = [\text{id}, f^\dagger]fJh$, meaning that $f^\dagger(Jh)$ satisfies the fixpoint equation for g^\dagger . Therefore, $f^\dagger(Jh) = g^\dagger$. ◀

► **Definition 19.** Let \mathbf{T} be a guarded monad, i.e. a monad with a guardedness predicate on $\text{Kl}(\mathbf{T})$. We say that \mathbf{T} is

1. a guarded iterative monad if $\text{Kl}(\mathbf{T})$ is a guarded iterative category;
2. a guarded Elgot monad if $\text{Kl}(\mathbf{T})$ has a guarded Conway operator $f \mapsto f^\dagger$, which is uniform wrt the obvious functor $\mathcal{C} \rightarrow \text{Kl}(\mathbf{T})$;

13:10 Coinductive Resumption Monads: Guarded Iterative and Guarded Elgot

3. an Elgot monad when it is totally guarded and a guarded Elgot monad.

Note that for an Elgot monad \mathbf{T} , $T0$ must always be inhabited because \mathbf{T} supports (unproductive) divergence $\perp = (\eta \text{ inr} : 1 \rightarrow T(0 + 1))^\dagger$.

► **Example 20.** Let us revisit Example 10.

1. Every monad under vacuous guardedness can be equipped with an iteration operator and is thus guarded iterative. Concretely, since $f : X \rightarrow Y \rangle X$ implies that

$$f = (X \xrightarrow{g} TY \xrightarrow{T \text{ inl}} T(Y + X))$$

for a suitable g , $f^\dagger = [\eta, f^\dagger]^*(T \text{ inl})g = g$. For Example 10 (1), therefore $f^\dagger = ! : X \rightarrow TY = 1$ if $Y \neq \emptyset$ and $f^\dagger = [] : \emptyset \rightarrow \emptyset$ if $Y = \emptyset$, and thus $X = \emptyset$.

2. The powerset monad \mathcal{P} is Elgot, because its Kleisli category (the category of relations) is enriched over complete partial orders, and hence supports f^\dagger as a least fixpoint of $[\eta, -]^*f$. This is inherited by \mathcal{P}^+ by restriction along the inclusion $\mathcal{P}^+ \hookrightarrow \mathcal{P}$. Explicitly, in the Kleisli category of \mathcal{P}^+ , for a guarded map $f : X \rightarrow Y \rangle X$, the map $f^\dagger : X \rightarrow Y$ sends x to the set

$$\{y \in Y \mid \exists n \in \mathbb{N}, (x_0, \dots, x_n) \in X^{n+1}. x = x_0 \wedge f(x_0) \ni \text{inr } x_1 \wedge \dots \wedge f(x_n) \ni \text{inl } y\}.$$

In this style of semantics we thus do not register the possibility of divergence i.e. whether there is a sequence $(x_0, x_1, \dots) \in X^\omega$ such that $\forall i \in \mathbb{N}. f(x_i) \ni \text{inr } x_{i+1}$. As a result, \mathcal{P}^+ is guarded Elgot.

3. Unlike its cousin, the *countable subdistribution monad* $\mathcal{D}X = \{d : X \rightarrow [0, 1] \mid \sum d \leq 1\}$, \mathcal{D}^+ is not Elgot (because $\mathcal{D}^+0 = 0$). However, as in the previous clause, it is guarded Elgot. Specifically, we obtain a guarded Conway operator for \mathcal{D}^+ by first restricting from the corresponding total guarded iteration operator for \mathcal{D} , calculated as a least fixed point, and then normalizing (guardedness ensures it is not zero) to obtain a distribution. Thus we do not record the probability of divergence. To see the need for normalization, consider the guarded Kleisli map $f : \mathbb{N} \rightarrow 1 \rangle \mathbb{N}$ where $f(n)$ gives $\text{inl } \star$ with probability

$$\frac{1}{2^n + 2} = \left(\frac{1}{2^{n+1}} \right) / \left(\frac{1}{2} + \frac{1}{2^n} \right)$$

and $\text{inr}(n + 1)$ with probability

$$1 - \frac{1}{2^n + 2} = \frac{2^n + 1}{2^n + 2} = \left(\frac{1}{2} + \frac{1}{2^{n+1}} \right) / \left(\frac{1}{2} + \frac{1}{2^n} \right).$$

Iterating f from 0 gives the probability $1/2^{n+2}$ of the transition sequence

$$0 \rightarrow 1 \rightarrow \dots \rightarrow n \rightarrow \star$$

So the probability of eventually reaching \star is $1/2$.

4. The writer monad $TX = A^* \times X$ does not support guarded iteration for the guardedness predicate defined in Example 10 (4) and for non-trivial A . For example, no $x : 1 \rightarrow T1 \cong A^*$ satisfies the fixpoint equation $x = ax$ for any $a \in A$. This can be remedied by extending TX to $A^* \times X + M$ where M is an inhabited left A^* -module, e.g. $M = 1$ (the initial one), or $M = A^\omega$ (the final one).
5. The monad $TX = \mathcal{P}^+(A^* \times X + A^\omega)$ for finite and infinite traces from Example 10 (5) supports the following iteration operator. For a guarded Kleisli map $f : X \rightarrow Y \rangle X$, the

map $f^\dagger: X \rightarrow TY$ sends x to the following set:

$$\begin{aligned} & \{ \text{inl}(m_0 + \dots + m_{n-1} + m, y) \mid \exists n \in \mathbb{N}, (x_0, \dots, x_n) \in X^{n+1}. \\ & \quad x_0 = x \\ & \quad \wedge f(x_0) \ni \text{inl}(m_0, \text{inr } x_1) \wedge \dots \wedge f(x_{n-1}) \ni \text{inl}(m_{n-1}, \text{inr } x_n) \\ & \quad \wedge f(x_n) \ni \text{inl}(m, \text{inl } y) \} \\ \cup & \{ \text{inr}(m_0 + \dots + m_{n-1} + m) \mid \exists n \in \mathbb{N}, (x_0, \dots, x_n) \in X^{n+1}. \\ & \quad x_0 = x \\ & \quad \wedge f(x_0) \ni \text{inl}(m_0, \text{inr } x_1) \wedge \dots \wedge f(x_{n-1}) \ni \text{inl}(m_{n-1}, \text{inr } x_n) \\ & \quad \wedge f(x_n) \ni \text{inr } m \} \\ \cup & \{ \text{inr}(m_0 + m_1 + \dots) \mid \exists (x_0, \dots) \in X^\omega. \\ & \quad x_0 = x \wedge \forall i \in \mathbb{N}. f(x_i) \ni \text{inl}(m_i, \text{inr } x_{i+1}) \}. \end{aligned}$$

This captures three possible scenarios (separated by the \cup operator):

- the fixpoint $f^\dagger(x)$ is unfolded n times resulting in an output of $y \in Y$; the actions $m_0, \dots, m_{n-1}, m \in A^*$ are collected along the run and concatenated;
- the fixpoint $f^\dagger(x)$ is unfolded n times and then hits an infinite trace $m \in A^\omega$; as a result, $f^\dagger(x)$ does not yield a value from Y , but it yields an infinite trace $m_0 + \dots + m_{n-1} + m \in A^\omega$ where $m_0, \dots, m_{n-1} \in A^*$ are collected along the run;
- the fixpoint $f^\dagger(x)$ is unfolded infinitely many times without ever reaching Y ; this yields an infinite trace $m_0 + m_1 + \dots \in A^\omega$ computed by concatenating the traces $m_i \in A^*$, which are collected along the run. The guardedness assumption on f is crucial here, because it ensures that each m_i is non-empty and hence the above infinite sum does indeed produce an infinite trace.

The resulting iteration operator is properly partial, and is computed neither as a least fixpoint nor as a unique fixpoint, even though the guardedness relation we postulate is the one standardly used in process algebra and guaranteeing uniqueness of fixpoint under strong bisimilarity [17]. The separating example is $x = ax + 1$, which has besides the canonical solution $x = a^* + a^\omega$ the solution $x = a^*$, ignoring the infinite trace.

5 The Coinductive Resumption Monad

In this section, we present our main technical contribution, stating that guarded Elgotness extends along the coalgebraic resumption monad transformer. It proves to be technically more advantageous to work more generally with *parametrized guarded Elgot monads*, which extend Uustalu's *parametrized monads* [23].

► **Definition 21** (Parametrized Guarded Elgot Monads). *A parametrized guarded Elgot monad is a functor from a co-Cartesian category \mathcal{C} to the category of guarded Elgot monads over \mathcal{C} . Equivalently (by uncurrying), a parametrized guarded Elgot monad is a bifunctor $\# : \mathcal{C} \times \mathcal{C} \rightarrow \mathcal{C}$, such that each $- \# W$ is a guarded Elgot monad, and for every $f : W \rightarrow W'$, $- \# f$ is a guarded Elgot monad morphism.*

Since every monad is a guarded Elgot monad under vacuous guardedness, parametrized guarded Elgot monads include all parametrized monads.

The main example of a parametrized guarded Elgot monad is as follows.

► **Example 22.** Given a guarded Elgot monad \mathbf{T} and an endofunctor H on the same co-Cartesian category \mathcal{C} , $X \# Y = T(X + HY)$ defines a parametrized guarded Elgot monad, with

13:12 Coinductive Resumption Monads: Guarded Iterative and Guarded Elgot

the guardedness predicate defined as follows: for $f: X \rightarrow T((Y + Z) + HW)$, $f: X \rightarrow Y \rangle Z$ in $\text{Kl}(- \# W)$ iff

$$X \xrightarrow{f} T((Y + Z) + HW) \cong T((Y + HW) + Z) \quad \text{is} \quad \text{inr-guarded in } \text{Kl}(\mathbf{T}).$$

We use the same Kleisli style notation for parametrized guarded Elgot monads as for the non-parametrized ones. Let us record the identities, directly implied by Definition 21, and which we use in the subsequent calculations.

$$\begin{aligned} (\eta_{X,Y} : X \rightarrow Z \# W)^* &= \text{id}_{X \# W} : X \# W \rightarrow X \# W, \\ (f : Y \rightarrow Z \# W)^*(\eta_{X,Y} : X \rightarrow X \# W) &= f : X \rightarrow Z \# W, \\ (f : Y \rightarrow Z \# W)^*(g : X \rightarrow Y \# W)^* &= (f^*g)^* : X \# W \rightarrow Z \# W, \\ (X \# (h : W \rightarrow W'))(\eta_{X,W} : X \rightarrow X \# W) &= \eta_{X,W'} : X \rightarrow X \# W', \\ (Y \# (h : W \rightarrow W'))(f : X \rightarrow Y \# W)^* &= \\ &= ((Y \# h)f)^*(Y \# h) : X \# W \rightarrow Y \# W', \\ (Y \# (h : W \rightarrow W'))(f : X \rightarrow (Y + X) \# W)^\dagger &= ((Y \# h)f)^\dagger : X \rightarrow Y \# W'. \end{aligned}$$

The first three equations here are the monad laws for $- \# W$ and the last three equations express the fact that $- \# h$ is a guarded Elgot monad morphism.

For the rest of the section we fix a parametrized guarded Elgot monad $\#$ and assume that the final coalgebras $F_{\#}X = \nu\gamma.X \# \gamma$ exist for all $X \in |\mathcal{C}|$. The following properties of $F_{\#}$ are previously shown by Uustalu [23].

► Proposition 23.

1. For every $f: X \rightarrow F_{\#}Y$ there is a unique $f^*: F_{\#}X \rightarrow F_{\#}Y$ such that

$$\begin{array}{ccc} F_{\#}X & \xrightarrow{\text{out}} & X \# F_{\#}X \\ f^* \downarrow & & \downarrow ((\text{out } f) \# f^*)^* \\ F_{\#}Y & \xrightarrow{\text{out}} & Y \# F_{\#}Y \end{array}$$

2. given $f: X \rightarrow Y \# F_{\#}(X + Y)$, there is unique $g: X \rightarrow F_{\#}Y$, such that

$$\begin{array}{ccc} X & \xrightarrow{f} & Y \# F_{\#}(Y + X) \\ g \downarrow & & \downarrow Y \# [\eta^\nu, g]^* \\ F_{\#}Y & \xrightarrow{\text{out}} & Y \# F_{\#}Y \end{array}$$

3. $F_{\#}$ forms a monad, whose unit $\eta^\#$ at X is

$$X \xrightarrow{\eta^{\text{inl}}} X \# F_{\#}X \xrightarrow{\text{out}^{-1}} F_{\#}X.$$

The Kleisli extension of $f: X \rightarrow F_{\#}Y$ is f^* .

5.1 Transferring Guarded Elgotness

We proceed to transfer iteration from $\#$ to $F_{\#}$.

► **Definition 24.** $F_{\#}$ is guarded as follows: $f: X \rightarrow Y \rangle Z$ when the composite

$$X \xrightarrow{f} F_{\#}(Y + Z) \xrightarrow{\text{out}} (Y + Z) \# F_{\#}(Y + Z)$$

is inr-guarded.

See [10] for a proof that this constitutes a guardedness predicate.

Note that when $\#$ is totally guarded then so is $F_{\#}$. Using Definition 24, for every $f: X \rightarrow F_{\#}(Y + X)$, we define $\diamond f = (\text{out } f)^{\dagger}: X \rightarrow Y \# F_{\#}(Y + X)$. The idea of this operator is as follows. Computing the iteration of $f: X \rightarrow F_{\#}(Y + X)$ w.r.t. $F_{\#}$ amounts to forming $\text{out } f: X \rightarrow (Y + X) \# F_{\#}(Y + X)$ first, which reveals two occurrences of X that must be iterated away. The first one occurs at the guarded position of the parametrized monad, and hence we can eliminate it by applying the iteration operator of $- \# F_{\#}(Y + X) -$ this is precisely the task of \diamond . The remaining second position of X occurs under $F_{\#}$, and can be eliminated by using the finality property of the latter.

► **Theorem 25.** $F_{\#}$ is a guarded Elgot monad with the iteration operator $(-)^{\dagger}$ characterized as follows: for $f: X \rightarrow Y \rangle X$, $f^{\dagger}: X \rightarrow F_{\#}Y$ is the unique morphism satisfying

$$\begin{array}{ccc} X & \xrightarrow{\diamond f} & Y \# F_{\#}(Y + X) \\ f^{\dagger} \downarrow & & \downarrow Y \#[\eta^{\nu}, f^{\dagger}]^* \\ F_{\#}Y & \xrightarrow{\text{out}} & Y \# F_{\#}Y \end{array}$$

Proof. Let us verify the relevant laws. Recall that by Proposition 17, we need not verify dinaturality.

- *fixpoint* is already shown in [10];
- *naturality*: given $f: X \rightarrow Y \rangle X$, $g: Y \rightarrow F_{\#}Z$, let us denote by h the morphism $[(F_{\#} \text{inl}) g, \eta^{\nu} \text{inr}] : Y + X \rightarrow Z \rangle X$, First we show that

$$\diamond(h^* f) = ((Z \# F_{\#} \text{inl}) \text{out } g)^*(Y \# h^*) \diamond f. \quad (1)$$

Indeed,

$$\begin{aligned} \diamond(h^* f) &= (\text{out } h^* f)^{\dagger} && // \text{ definition of } \diamond \\ &= ((\text{out } h)^*(Y \# h^*) \text{out } f)^{\dagger} \\ &= ((Z \# F_{\#} \text{inl}) \text{out } g)^*((Y \# h^*) \text{out } f)^{\dagger} && // \text{ naturality of } (-)^{\dagger} \\ &= ((Z \# F_{\#} \text{inl}) \text{out } g)^*(Y \# h^*) \diamond f. \end{aligned}$$

Then

$$\begin{aligned} \text{out } g^* f^{\dagger} &= (\text{out } g)^*(Y \# g^*) (Y \# [\eta^{\nu}, f^{\dagger}]^*) \diamond f && // \text{ definition of } (-)^{\dagger} \\ &= (\text{out } g)^*(Y \# (g^*[\eta^{\nu}, f^{\dagger}]^*)) \diamond f \\ &= (\text{out } g)^*(Y \# [\eta^{\nu}, g^* f^{\dagger}]^* [(F_{\#} \text{inl}) g, \eta^{\nu} \text{inr}]^*) \diamond f \\ &= (\text{out } g)^*(Y \# [\eta^{\nu}, g^* f^{\dagger}]^* h^*) \diamond f \\ &= (Z \# [\eta^{\nu}, g^* f^{\dagger}]^*) ((Z \# F_{\#} \text{inl}) \text{out } g)^*(Y \# h^*) \diamond f && // (1) \\ &= (Z \# [\eta^{\nu}, g^* f^{\dagger}]^*) \diamond(h^* f). \end{aligned}$$

This entails the requisite equality $(h^* f)^{\dagger} = g^* f^{\dagger}$, by the uniqueness property of $(h^* f)^{\dagger}$.

- *codiagonal*: let $f: X \rightarrow Y \rangle X \rangle X$. It suffices to check that

$$\text{out } f^{\dagger\dagger} = (Y \# [\eta^{\nu}, f^{\dagger\dagger}]^*) \diamond (F_{\#}[\text{id}, \text{inr}] f)$$

13:14 Coinductive Resumption Monads: Guarded Iterative and Guarded Elgot

The proof runs as follows:

$$\begin{aligned}
\text{out } f^{\ddagger\ddagger} &= (Y \# [\eta^\nu, f^{\ddagger\ddagger}]^*) \diamond f^{\ddagger} && // \text{ definition of } (-)^{\ddagger} \\
&= (Y \# [\eta^\nu, f^{\ddagger\ddagger}]^*) (\text{out } f^{\ddagger})^\dagger && // \text{ definition of } \diamond \\
&= (Y \# [\eta^\nu, f^{\ddagger\ddagger}]^*) ((Y + X) \# [\eta^\nu, f^{\ddagger}]^* \diamond f)^\dagger && // \text{ definition of } (-)^{\ddagger} \\
&= (Y \# [\eta^\nu, f^{\ddagger\ddagger}]^* [\eta^\nu, f^{\ddagger}]^*) (\text{out } f)^{\dagger\dagger} \\
&= (Y \# [[\eta^\nu, f^{\ddagger\ddagger}], [\eta^\nu, f^{\ddagger}]^* f^{\ddagger}]^*) (\text{out } f)^{\dagger\dagger} \\
&= (Y \# [[\eta^\nu, f^{\ddagger\ddagger}], f^{\ddagger}]^*) (\text{out } f)^{\dagger\dagger} && // \text{ fixpoint for } (-)^{\ddagger} \\
&= (Y \# [\eta^\nu, f^{\ddagger\ddagger}]^* F_\#[\text{id}, \text{inr}]) (\text{out } f)^{\dagger\dagger} \\
&= (Y \# [\eta^\nu, f^{\ddagger\ddagger}]^*) (Y \# F_\#[\text{id}, \text{inr}]) \\
&\quad (([\text{id}, \text{inr}] \# F_\#((Y + X) + X)) \text{out } f)^\dagger && // \text{ codiagonal for } (-)^{\ddagger} \\
&= (Y \# [\eta^\nu, f^{\ddagger\ddagger}]^*) (([\text{id}, \text{inr}] \# F_\#[\text{id}, \text{inr}]) \text{out } f)^\dagger \\
&= (Y \# [\eta^\nu, f^{\ddagger\ddagger}]^*) (\text{out } F_\#[\text{id}, \text{inr}]f)^\dagger \\
&= (Y \# [\eta^\nu, f^{\ddagger\ddagger}]^*) \diamond (F_\#[\text{id}, \text{inr}]f) && // \text{ definition of } \diamond
\end{aligned}$$

- *uniformity*: assume $f: X \rightarrow Y \triangleright X$, $g: Z \rightarrow Y \triangleright Z$, $h: Z \rightarrow X$ and $f h = F_\#(\text{id} + h) g$. The latter entails $(\text{out } f) h = ((\text{id} + h) \# F_\#(\text{id} + h)) \text{out } g$, hence, by uniformity of $(-)^{\ddagger}$,

$$(\text{out } f)^\dagger h = ((Y \# F_\#(\text{id} + h)) \text{out } g)^\dagger. \quad (2)$$

We then have

$$\begin{aligned}
\text{out } f^{\ddagger} h &= (Y \# [\eta^\nu, f^{\ddagger}]^*) (\diamond f) h && // \text{ definition of } (-)^{\ddagger} \\
&= (Y \# [\eta^\nu, f^{\ddagger}]^*) (\text{out } f)^\dagger h && // \text{ definition of } \diamond \\
&= (Y \# [\eta^\nu, f^{\ddagger}]^*) ((Y \# F_\#(\text{id} + h)) \text{out } g)^\dagger && // (2) \\
&= (Y \# [\eta^\nu, f^{\ddagger} h]^*) (\text{out } g)^\dagger \\
&= (Y \# [\eta^\nu, f^{\ddagger} h]^*) \diamond g && // \text{ definition of } \diamond
\end{aligned}$$

We thus obtained that $f^{\ddagger} h$ satisfies the fixpoint equation for g^{\ddagger} , hence $f^{\ddagger} h = g^{\ddagger}$. ◀
Recall that $T_H^\nu = F_\#$ for $X \# Y = T(X + HY)$.

► **Corollary 26.** *Given a guarded Elgot monad \mathbf{T} , then \mathbf{T}_H^ν is also guarded Elgot with the requisite structure obtained from the parametrized guarded Elgot monad $X \# Y = T(X + HY)$.*

5.2 Free Extensions of Guarded Elgot Monads

We proceed to apply the results of the previous section under $X \# Y = T(X + HY)$.

► **Lemma 27.** *Let \mathbf{T} be a guarded monad.*

1. *The natural transformation*

$$HX \xrightarrow{\eta \text{ inr}(H\eta^\nu)} T(X + HT_H^\nu X) \xrightarrow{\text{out}^{-1}} T_H^\nu X$$

is guarded.

2. *The natural transformation*

$$TX \xrightarrow{T \text{ inl}} T(X + HT_H^\nu X) \xrightarrow{\text{out}^{-1}} T_H^\nu X$$

is a guarded Elgot monad morphism.

Proof. The first clause is obvious by definition. For the second clause, we need to check that the relevant morphism preserves guarded iteration, that is, given $f : X \rightarrow Y \triangleright X$, we need to show that

$$\text{out}^{-1}(T \text{ inl}) f^\dagger = (\text{out}^{-1}(T \text{ inl}) f)^\ddagger.$$

By definition of $(-)^{\ddagger}$, equivalently, we prove the equation

$$(T \text{ inl}) f^\dagger = T(\text{id} + [\eta^\nu, (T \text{ inl}) f^\dagger]^*) \diamond (\text{out}^{-1}(T \text{ inl}) f).$$

Indeed,

$$\begin{aligned} & T(\text{id} + [\eta^\nu, (T \text{ inl}) f^\dagger]^*) \diamond (\text{out}^{-1}(T \text{ inl}) f) \\ &= T(\text{id} + [\eta^\nu, (T \text{ inl}) f^\dagger]^*) (T(\text{inl} + \text{id}) f)^\dagger && // \text{ definition of } \diamond \\ &= T(\text{id} + [\eta^\nu, (T \text{ inl}) f^\dagger]^*) (T \text{ inl}) f^\dagger && // \text{ naturality} \\ &= (T \text{ inl}) f, \end{aligned}$$

as desired. \blacktriangleleft

► **Theorem 28.** *In the category of guarded Elgot monads, \mathbf{T}_H^ν , provided it exists, is a free extension of \mathbf{T} by H in the sense of Definition 2, that is, for every guarded Elgot monad \mathbf{S} , a guarded Elgot monad morphism $\xi : \mathbf{T} \rightarrow \mathbf{S}$ and a guarded natural transformation $\sigma : H \rightarrow \mathbf{S}$, there exists a guarded Elgot monad morphism $\zeta : \mathbf{T}_H^\nu \rightarrow \mathbf{S}$ uniquely characterized by the following commutative diagram*

$$\begin{array}{ccccccc} T & \xrightarrow{T \text{ inl}} & T(\text{Id} + HT_H^\nu) & \xrightarrow{\text{out}^{-1}} & T_H^\nu & \xleftarrow{\text{out}^{-1}} & T(\text{Id} + HT_H^\nu) & \xleftarrow{\eta \text{ inr } H \eta^\nu} & H \\ & \searrow \xi & & & \downarrow \zeta & & & & \swarrow \sigma \\ & & & & S & & & & \end{array}$$

in the obvious category of natural transformations. Concretely, every $\zeta_X : T_H^\nu X \rightarrow SX$ has the form $(T_H^\nu X \xrightarrow{\xi \text{ out}} S(X + HT_H^\nu X) \xrightarrow{[\eta \text{ inl}, (S \text{ inr}) \sigma]^*} S(X + T_H^\nu X))^\dagger$.

Proof. By Lemma 27 the candidate monad morphism $\mathbf{T} \rightarrow \mathbf{T}_H^\nu$ and the candidate natural transformation $H \rightarrow \mathbf{T}_H^\nu$ are guarded. The trickiest part of the claim is the fact that \mathbf{T}_H^ν is indeed a guarded Elgot monad, which is shown in Theorem 25. Let us verify that ζ is a guarded monad morphism. Suppose that $f : X \rightarrow T_H^\nu(Y + X)$ is inr-guarded and show that $\zeta f : X \rightarrow S(Y + X)$ is inr-guarded. We have

$$\begin{aligned} \zeta f &= [\eta, \zeta]^* [\eta \text{ inl}, (S \text{ inr}) \sigma]^* \xi \text{ out } f && // \text{ fixpoint} \\ &= [\eta, \zeta^* \sigma]^* \xi \text{ out } f, \end{aligned}$$

which can be presented as

$$\begin{aligned} X & \xrightarrow{\text{out } f} T((Y + X) + HT_H^\nu(Y + X)) \cong T((Y + HT_H^\nu(Y + X)) + X) \\ & \xrightarrow{[[\eta \text{ inl}, \zeta^* \sigma], \eta \text{ inr}]^*} S(Y + X). \end{aligned}$$

The composite morphism in the upper row is inr-guarded by definition. We will be done by **(cmp)** if we show that the morphism $[\eta \text{ inl}, \zeta^* \sigma]$ occurring in the lower row is inr-guarded. By **(trv)** and **(par)** this amounts to showing that $\zeta^* \sigma : HT_H^\nu(Y + X) \rightarrow S(Y + X)$ is

inr-guarded. Indeed, σ is id-guarded by definition, hence $\zeta^*\sigma$ is id-guarded by **(cmp)**, which weakens to inr-guardedness by Proposition 8.

The remaining calculations are the same as in previous work [9, Theorem 8.3] where the analogous statement was shown in the unguarded case. \blacktriangleleft

Note that the initial guarded Elgot monad is the identity monad under vacuous guardedness. Using Proposition 5 we obtain

► **Corollary 29.** *Given a guarded Elgot monad \mathbf{T} and an endofunctor H , \mathbf{T}_H^ν is the coproduct, in the category of guarded Elgot monads, of \mathbf{T} and $\nu\gamma.(-+H\gamma)$, provided the latter exists.*

6 Conclusions and Further Work

We introduced guarded Elgot monads as a common generalization of Elgot monads and guarded iterative monads previously studied in the literature. We propose to use them as a yardstick for analyzing sophisticated notions of iteration when the iteration operator is neither total nor a unique solution of the corresponding fixpoint equation. Situations of this kind indeed occur in practice, e.g. in process semantics wrt infinite trace equivalence (Example 20 (5)). Moreover, guarded Elgotness tends to propagate along monad transformers, which leads to further examples. We explored one such monad transformer, receiving as an input a monad \mathbf{T} and a functor H and returning a monad \mathbf{T}_H^ν of possibly non-terminating processes under strong bisimilarity, with side-effects by \mathbf{T} and with actions by H . Our main theorem shows that for a guarded Elgot monad \mathbf{T} , \mathbf{T}_H^ν is canonically guarded Elgot, and more specifically it can be characterized as a free extension of \mathbf{T} by H in the category of all guarded Elgot monads.

The monad transformer $\mathbf{T} \mapsto \mathbf{T}_H^\nu$ is particularly important because the semantic domain it generates is subject to (coalgebraic) strong bisimilarity, which is arguably the finest semantic equivalence on processes. We thus hope to obtain further interesting generic process equivalences, most importantly infinite trace equivalence, in a principled fashion, as quotients of \mathbf{T}_H^ν under suitably defined iteration-congruences. We plan to explore connections between the outlined approach to characterizing process equivalences and universal characterizations of such equivalences, such as the final coalgebra characterization of finite trace equivalence given in [4].

References

- 1 J. Adámek, R. Börger, S. Milius, and J. Velebil. Iterative algebras: How iterative are they? *Theory Appl. Cat.*, 19:61–92, 2008.
- 2 Stephen Bloom and Zoltán Ésik. *Iteration theories: the equational logic of iterative processes*. Springer, 1993.
- 3 F. Borceux. *Handbook of Categorical Algebra 1*. Cambridge University Press, 1994.
- 4 N. Bowler, P.B. Levy, and G.D. Plotkin. Initial algebras and final coalgebras consisting of nondeterministic finite trace strategies. In *Proceedings, 34th Conference on the Mathematical Foundations of Programming Semantics*, volume 341 of *ENTCS*, pages 23–44, 2018.
- 5 Venanzio Capretta. General recursion via coinductive types. *Log. Meth. Comput. Sci.*, 1(2), 2005.
- 6 Pietro Cenciarelli and Eugenio Moggi. A syntactic approach to modularity in denotational semantics. In *Category Theory and Computer Science, CTCS 1993*, 1993.
- 7 Sergey Goncharov, Stefan Milius, and Christoph Rauch. Complete Elgot Monads and Coalgebraic Resumptions. In *Mathematical Foundations of Programming Semantics, MFPS 2016*, volume 325 of *ENTCS*, pages 147–168. Elsevier, 2016.

- 8 Sergey Goncharov and Lutz Schröder. Guarded Traced Categories. In Christel Baier and Ugo Dal Lago, editors, *Proc. 21th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2018)*, volume 10803 of *LNCS*, pages 313–330. Springer, 2018.
- 9 Sergey Goncharov, Lutz Schröder, Christoph Rauch, and Julian Jakob. Unguarded Recursion on Coinductive Resumptions. *Logical Methods in Computer Science*, 14(3), 2018.
- 10 Sergey Goncharov, Lutz Schröder, Christoph Rauch, and Maciej Piróg. Unifying Guarded and Unguarded Iteration. In Javier Esparza and Andrzej Murawski, editors, *Foundations of Software Science and Computation Structures, FoSSaCS 2017*, volume 10203 of *LNCS*, pages 517–533. Springer, 2017.
- 11 Masahito Hasegawa. *Models of Sharing Graphs: A Categorical Semantics of Let and Letrec*. Springer, 1999.
- 12 Masahito Hasegawa. Recursion from Cyclic Sharing: Traced Monoidal Categories and Models of Cyclic Lambda Calculi. In *Typed Lambda Calculi and Applications, TLCA 1997*, volume 1210 of *LNCS*, pages 196–213. Springer, 1997.
- 13 Martin Hyland, Paul Levy, Gordon Plotkin, and John Power. Combining algebraic effects with continuations. *Theoret. Comput. Sci.*, 375(1-3):20–40, 2007.
- 14 Martin Hyland, Gordon Plotkin, and John Power. Combining Computational Effects: Commutativity & Sum. In *TCS'02*, volume 223, pages 474–484. Kluwer, 2002.
- 15 Martin Hyland, Gordon Plotkin, and John Power. Combining effects: Sum and tensor. *Theoret. Comput. Sci.*, 357(1-3):70–99, 2006.
- 16 Stefan Milius. Completely iterative algebras and completely iterative monads. *Inf. Comput.*, 196(1):1–41, 2005.
- 17 R. Milner. *Communication and concurrency*. Prentice-Hall, 1989.
- 18 Eugenio Moggi. Notions of Computation and Monads. *Inf. Comput.*, 93:55–92, 1991.
- 19 Maciej Piróg and Jeremy Gibbons. The Coinductive Resumption Monad. In *Mathematical Foundations of Programming Semantics, MFPS 2014*, volume 308 of *ENTCS*, pages 273–288, 2014.
- 20 Maciej Piróg and Jeremy Gibbons. Monads for Behaviour. In *Mathematical Foundations of Programming Semantics, MFPS 2013*, volume 298 of *ENTCS*, pages 309–324, 2015.
- 21 A. W. Roscoe. Unbounded nondeterminism in CSP. Technical Report PRG-67, Oxford University Computing Laboratory, July 1988. in *Two papers on CSP*, Also appeared in *Journal of Logic and Computation*, Vol 3, No 2 pp131-172 (1993). URL: <http://www.cs.ox.ac.uk/people/bill.roscoe/publications/28.ps>.
- 22 Alex Simpson and Gordon Plotkin. Complete axioms for categorical fixed-point operators. In *Logic in Computer Science, LICS 2000*, pages 30–41, 2000.
- 23 Tarmo Uustalu. Generalizing Substitution. *ITA*, 37(4):315–336, 2003.

Decomposing Comonad Morphisms

Danel Ahman 

Faculty of Physics and Mathematics, University of Ljubljana, Slovenia
danel.ahman@fmf.uni-lj.si

Tarmo Uustalu 

Department of Computer Science, Reykjavik University, Iceland
Dept. of Software Science, Tallinn University of Technology, Estonia
tarmo@ru.is

Abstract

The analysis of set comonads whose underlying functor is a container functor in terms of directed containers makes it a simple observation that any morphism between two such comonads factors through a third one by two comonad morphisms, whereof the first is identity on shapes and the second is identity on positions in every shape. This observation turns out to generalize into a much more involved result about comonad morphisms to comonads whose underlying functor preserves Cartesian natural transformations to itself on any category with finite limits. The bijection between comonad coalgebras and comonad morphisms from costate comonads thus also yields a decomposition of comonad coalgebras.

2012 ACM Subject Classification Theory of computation → Logic; Theory of computation → Categorical semantics

Keywords and phrases container functors (polynomial functors), container comonads, comonad morphisms and comonad coalgebras, cofunctors, lenses

Digital Object Identifier 10.4230/LIPIcs.CALCO.2019.14

Funding *Danel Ahman*: This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-17-1-0326.

Tarmo Uustalu: Supported by the Icelandic Research Fund project grant no. 196323-051 and the Estonian Ministry of Education and Research institutional research grant no. IUT33-13.

Acknowledgements We thank our anonymous reviewers for very useful remarks.

1 Introduction

Containers of Abbott et al. [1] are a representation of a wide class of set functors (that one can use as parameterized datatypes) in terms of shapes and positions. Those set functors that enjoy this representation are called container functors. In joint work with Chapman [3], we found that container functors with a comonad structure can be characterized as interpretations of containers with corresponding additional structure, which we called directedness. In a directed container, every position in a shape determines another shape (its subshape), every shape has a designated root position, and positions in a subshape can be translated to the original shape. Remarkably, as we only noticed later [5], the category of directed containers is equivalent to the opposite of the category of small categories and cofunctors. Cofunctors were introduced by Aguiar [2]; they send objects from the target category to the source category, but maps from the source category to the target category.

Motivated by this equivalence, in this paper, we first show that an analogue of the full image factorization of functors holds for directed container morphisms: any directed container morphism decomposes into two whereby the first is identity on shapes and the second is identity on positions in every shape. Since the interpretation functor from the category of directed containers to the category of set comonads is fully-faithful, this immediately gives also a factorization of container comonad morphisms.



© Danel Ahman and Tarmo Uustalu;

licensed under Creative Commons License CC-BY

8th Conference on Algebra and Coalgebra in Computer Science (CALCO 2019).

Editors: Markus Roggenbach and Ana Sokolova; Article No. 14; pp. 14:1–14:19

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Then we ask if a similar decomposition is possible for general comonads on general categories. We show that only pullbacks and a terminal object (i.e., finite limits) are needed in order to formulate suitable substitutes for the notions of identity-on-shapes container morphism and identity-on-positions-in-every-shape container morphism for general natural transformations, and to obtain the factorization of general comonad morphisms.

That this decomposition is possible at this level of generality is nice, we find, since an alternative would have been to switch from containers to polynomials [7, 8]. At the basic level, containers and polynomials can be considered each other’s notational variants, but the concepts of polynomials and polynomial functors scale to general categories with pullbacks [18]. However, already the definition of the polynomial analogue of the concept of directed container is complicated (we spelled it out in [3]), not to speak of the definition of the interpretation functor for it, or any proofs, so they are not the easiest to work with. The shapely types of Jay and Cockett [12, 11] are a lighter concept, but we did not need even those for our purpose.

The paper is organized as follows. In Section 2, we review containers and directed containers, including the equivalence of the category of directed containers to the opposite of the category of small categories and cofunctors. In Section 3, we describe our factorization of directed container morphisms or, which is the same, container comonad morphisms. In Section 4, we generalize this factorization to categories with finite limits. In Section 4 we also apply our results to the factorization of comonad coalgebras. We sum up in Section 5.

2 Preliminaries: containers and directed containers

We begin with a review of containers [1] and directed containers [3]. As noted above, containers are a representation for a certain class of set functors. Directed containers characterize, by additional structure on containers, those container functors that carry comonad structure.¹

A *container* comprises a set S (of shapes) and, for any shape $s : S$, a set $P s$ (of positions in shape s). A *directed container* is a container (S, P) equipped with three maps

- $\downarrow : (\Sigma s : S. P s) \rightarrow S$ (the subshape corresponding to a position in a shape),
 - $\circ : \Pi_{s:S}. P s$ (the root position in a given shape), and
 - $\oplus : \Pi_{s:S}. (\Sigma p : P s. P (s \downarrow p)) \rightarrow P s$ (translation of a position in a position’s subshape)
- satisfying the following five equations:

$$\begin{aligned} s \downarrow \circ_s &= s & s \downarrow (p \oplus_s p') &= (s \downarrow p) \downarrow p' \\ p \oplus_s \circ_{s \downarrow p} &= p & \circ_s \oplus_s p &= p & (p \oplus_s p') \oplus_s p'' &= p \oplus_s (p' \oplus_{s \downarrow p} p'') \end{aligned}$$

The 4th and 5th equations type because the 1st and 2nd hold. We note that the data and equations of a directed container are like those of a set, a monoid, and a right action of the monoid on the set, modulo the presence of the “minor” (subscripted) arguments and the dependent typing. In particular, if $P s$, \circ_s , and $p \oplus_s p'$ do not actually depend on s , then we indeed have a set, a monoid, and a right action of the monoid.

A container (S, P) defines a set functor $\llbracket S, P \rrbracket^c = D$, called its *interpretation*, by

$$D X = \Sigma s : S. P s \Rightarrow X$$

¹ In what follows, subscript arguments of operations are “minor” arguments that can typically be inferred from the subsequent arguments. We generally write \rightarrow for homsets, and \Rightarrow for internal homs (exponential objects). In this and the next section, where we work in **Set**, this plays no role, but we still use the notation for conceptual clarity.

Given a directed container structure $(\downarrow, \circ, \oplus)$ on (S, P) , D obtains a comonad structure:

$$\varepsilon_X(s, v) = v \circ_s \quad \delta_X(s, v) = (s, \lambda p. (s \downarrow p, \lambda p'. v (p \oplus_s p')))$$

We call the comonad $\llbracket S, P, \downarrow, \circ, \oplus \rrbracket^{\text{dc}} = (D, \varepsilon, \delta)$ the *interpretation* of $(S, P, \downarrow, \circ, \oplus)$.

Any comonad structure (ε, δ) on a set functor D that is the interpretation of some container (S, P) (i.e., $DX = \Sigma s : S. P s \Rightarrow X$) arises from a directed container structure $(\downarrow, \circ, \oplus)$ on (S, P) . In fact, directed container structures on (S, P) and comonad structures on D are in a bijection. Given a comonad structure (ε, δ) , the corresponding directed container structure is defined by

$$\circ_s = \varepsilon_{P s}(s, \text{id}) \quad s \downarrow p = \text{fst}(\text{snd}(\delta_{P s}(s, \text{id}))) p \quad p \oplus_s p' = \text{snd}(\text{snd}(\delta_{P s}(s, \text{id}))) p p'$$

The following are some most prominent examples of directed containers with the corresponding comonads:

- Taking S to be any set, $P s = 1$, $s \downarrow * = s$, $\circ_s = *$, $* \oplus_s * = *$, we get the *coreader comonad* defined by $DX = S \times X \cong \Sigma s : S. 1 \Rightarrow X$, $\varepsilon(s, x) = x$, $\delta(s, x) = (s, (s, x))$.
- Taking S to be any set, $P s = S$, $s \downarrow s' = s'$, $\circ_s = s$, $s' \oplus_s s'' = s''$, we get the *costate comonad* (also called the *array comonad* [16]) defined by $DX = S \times (S \Rightarrow X) \cong \Sigma s : S. S \Rightarrow X$, $\varepsilon(s, v) = v s$, $\delta(s, v) = (s, \lambda s'. (s', v))$.
- Choosing $S = 1$, $P * = \mathbb{N}$, $* \downarrow i = *$, $\circ_* = 0$, $i \oplus_* j = i + j$, we obtain the *streams-with-suffixes comonad* defined by $DX = X^\omega \cong \Sigma s : 1. \mathbb{N} \Rightarrow X$, $\varepsilon(x_0, x_1, \dots) = x_0$, $\delta(x_0, x_1, \dots) = ((x_0, x_1, \dots), (x_1, x_2, \dots), \dots)$.
- Choosing $S = \mathbb{N}$, $P n = [0..n]$, $n \downarrow i = s - i$, $\circ_n = 0$, $i \oplus_n j = i + j$ gives us the *nonempty-lists-with-suffixes comonad* defined by $DX = X^+ \cong \Sigma n : \mathbb{N}. [0..n] \Rightarrow X$, $\varepsilon(x_0, x_1, \dots, x_n) = x_0$, $\delta(x_0, x_1, \dots, x_n) = ((x_0, x_1, \dots, x_n), (x_1, x_2, \dots, x_n), \dots, (x_n))$.
- Take S to be the set of all bars where a *bar* (through the binary fan) is a finite set b of lists over $2 = \{0, 1\}$ such that any stream over 2 has exactly one prefix in b . Take $P b$ to be the set of all lists u over 2 that are a prefix of some list w in b . (A bar cuts a finite binary tree out of the infinite binary tree by establishing the positions of its leaves). Let $b \downarrow u = \{v \mid u \cdot v \in b\}$, $\circ_b = ()$, $u \oplus_b v = u \cdot v$ (the empty list resp. concatenation of lists). This gives us the *labelled-finite-binary-trees comonad*. The counit extracts the label of the root node of the given tree. The comultiplication replaces the label of each node with the subtree rooted by that node.

Other useful examples of directed containers are obtained by constructions corresponding to the coproduct and product of two comonads, the cofree comonad on a functor, and compatible compositions of comonads (for all these, there are corresponding constructions of directed containers) and zipper datatypes (for those, there is a construction, called focussing, of turning any container (S, P) into a directed container whose shape set is $\Sigma s : S. P s$, i.e., its shapes are shapes of the given container together with a focus position) [3, 4].

A *morphism* between two containers (S, P) and (S', P') is given by maps $t : S \rightarrow S'$ (the shape map) and $q : \Pi_{s:S}. P'(ts) \rightarrow P s$ (the position map). A *morphism* between two directed containers $(S, P, \downarrow, \circ, \oplus)$ and $(S', P', \downarrow', \circ', \oplus')$ is a morphism (t, q) between the underlying containers satisfying the following equations

$$t(s \downarrow q_s p) = t s \downarrow' p \quad \circ_s = q_s \circ'_{t s} \quad q_s p \oplus_s q_{s \downarrow q_s p} p' = q_s (p \oplus'_{t s} p')$$

14:4 Decomposing Comonad Morphisms

Analogously to the interpretation of containers, a morphism (t, q) between containers (S, P) and (S', P') defines a natural transformation $\llbracket t, q \rrbracket^c = \tau$ between their interpretations $\llbracket S, P \rrbracket^c = D$ and $\llbracket S', P' \rrbracket^c = D'$ (the *interpretation* of (t, q)) by

$$\tau_X(s, v) = (t s, v \circ q_s)$$

Also, analogously to the interpretation of directed containers, if (t, q) is a morphism between directed containers $(S, P, \downarrow, \circ, \oplus)$ and $(S', P', \downarrow', \circ', \oplus')$, then τ is a comonad morphism between $\llbracket S, P, \downarrow, \circ, \oplus \rrbracket^{dc} = (D, \varepsilon, \delta)$ and $\llbracket S', P', \downarrow', \circ', \oplus' \rrbracket^{dc} = (D', \varepsilon', \delta')$; we define $\llbracket t, q \rrbracket^{dc} = \tau$.

Any natural transformation τ between the interpretations D and D' of two containers (S, P) and (S', P') is an interpretation of a unique container morphism, namely (t, q) where

$$t s = \text{fst}(\tau_{P s}(s, \text{id})) \quad q_s p = \text{snd}(\tau_{P s}(s, \text{id})) p$$

Furthermore, if τ is a comonad morphism between the interpretations (D, ε, δ) and $(D', \varepsilon', \delta')$ of two directed containers $(S, P, \downarrow, \circ, \oplus)$ and $(S', P', \downarrow', \circ', \oplus')$, then (t, q) is a directed container morphism interpreting to τ .

Some examples of directed container morphisms are the following:

- Let $(S, P, \downarrow, \circ, \oplus)$ and $(S', P', \downarrow', \circ', \oplus')$ be the directed containers for the costate comonad for S and coreader comonad for S , respectively. Take $t s = s$, $q_s * = s$. This corresponds to the comonad morphism $\tau_X : S \times (S \Rightarrow X) \rightarrow S \times X$ defined by $\tau(s, v) = (s, v s)$.
- Let $(S, P, \downarrow, \circ, \oplus)$ and $(S', P', \downarrow', \circ', \oplus')$ be the directed containers for the nonempty lists comonad and the streams comonad, respectively. Take $t n = *$ and $q_n i = \min(i, n)$. This corresponds to the comonad morphism $\tau_X : X^+ \rightarrow X^\omega$ defined by $\tau(x_0, x_1, \dots, x_n) = (x_0, x_1, \dots, x_n, x_n, \dots)$ (i.e., nonempty lists are padded out to streams).
- Let both $(S, P, \downarrow, \circ, \oplus)$ and $(S', P', \downarrow', \circ', \oplus')$ be the directed container for the nonempty lists comonad. Let $t n = n \div 2$ and $q_n i = 2 * i$. This corresponds to the comonad morphism $\tau(x_0, x_1, \dots, x_n) = (x_0, x_2, \dots, x_{2*(n \div 2)})$ (i.e., every other element of a given nonempty list is dropped).
- Let $(S, P, \downarrow, \circ, \oplus)$ and $(S', P', \downarrow', \circ', \oplus')$ be the directed containers for the nonempty lists comonad and the labelled finite binary trees comonad. Let $t n = \{w \in 2^* \mid |w| = n\}$ and $q_n u = |u|$. This corresponds to the comonad morphism sending a nonempty list $x s$ to a labelled finite binary tree whose list of labels along any path is $x s$ (so all paths have same length).
- Let $(S, P, \downarrow, \circ, \oplus)$ and $(S', P', \downarrow', \circ', \oplus')$ be the directed containers for the labelled finite binary trees comonad and nonempty lists comonad. Let $t b$ be the length of the unique prefix in b of the stream 0^ω , i.e., the unique n such that $0^n \in b$. Let $q_b i = 0^i$. This directed container morphism (t, q) then represents the comonad morphism that maps a labelled finite binary tree to the non-empty list of labels along its leftmost path.

Containers and container morphisms form a monoidal category **Cont** (with a suitable container composition monoidal structure), and the interpretation of containers is a fully-faithful monoidal functor from **Cont** to **[Set, Set]** (with the functor composition monoidal structure). Analogously, directed containers and directed container morphisms form a category **DCont**, and the interpretation of directed containers is fully-faithful functor from **DCont** to **Comonad(Set)**. In fact, **DCont** is isomorphic to the category **Comonoid(Cont)** and is the pullback in **CAT** of $U : \mathbf{Comonad}(\mathbf{Set}) \rightarrow [\mathbf{Set}, \mathbf{Set}]$ along $\llbracket - \rrbracket^c : \mathbf{Cont} \rightarrow [\mathbf{Set}, \mathbf{Set}]$.

In a sequel [5] to the first directed container work [3], we related directed containers to small categories. It turns out that directed containers are in a bijection up to isomorphism

with small categories. Specifically, given a directed container $(S, P, \downarrow, \mathbf{o}, \oplus)$, the corresponding small category is obtained as follows. The set of objects is S . The set of maps with domain $s : S$ is $P s$, which means that the total set of maps is $\bar{P} = \Sigma s : S. P s$ and the domain of a map $(s, p) : \bar{P}$ is $\text{src } p = s$. The codomain of a map $(s, p) : \bar{P}$ is $\text{tgt } p = s \downarrow p$. The identity map on an object s is $\text{id}_s = (s, \mathbf{o}_s)$ and the 1st directed container equation ensures that its codomain is $s \downarrow \mathbf{o}_s = s$, as required. A map (s, p) can only be composed with a map (s', p') , if $s \downarrow p = s'$, in which case the composition is $(s, p); (s', p') = (s, p \oplus_s p')$. By the 2nd directed container equation the codomain of this map is $s \downarrow (p \oplus_s p') = (s \downarrow p) \downarrow p'$, as required. The 3rd to the 5th equations then ensure that composition is unital and associative.

Of the above examples, the coreader comonad for S corresponds to the free category on a set of objects S , i.e., the discrete category (the only maps are the identity maps for every object). The costate comonad for S corresponds to the cofree category on a set of objects S , i.e., the codiscrete category (there is exactly one map between any two objects).

Although directed containers are in a bijection up to isomorphism with small categories, the category of directed containers is *not* equivalent to the category of small categories. The reason is that directed container morphisms are nothing like functors between small categories. Instead, they correspond to what Aguiar [2] has termed cofunctors, but with the source and target categories swapped.

A *cofunctor* between small categories $(S', \bar{P}', \text{src}', \text{tgt}', \text{id}', ;')$ and $(S, \bar{P}, \text{src}, \text{tgt}, \text{id}, ;)$ is given by two maps $t : S \rightarrow S'$ (the object map) and $\bar{q} : (\Sigma s : S. \Sigma p : \bar{P}'. t s = \text{src } p) \rightarrow \bar{P}$ (the morphism map) satisfying $\text{src } (\bar{q}(s, p)) = s$ and the following equations:

$$t(\text{tgt } (\bar{q}(s, p))) = \text{tgt}' p \quad \text{id}_s = \bar{q}(s, \text{id}'_{t s}) \quad \bar{q}(s, p); \bar{q}(\text{tgt } (\bar{q}(s, p)), p') = \bar{q}(s, p; ' p')$$

While a functor maps objects and maps of the source category to those in the target category, a cofunctor's object map is from the target category to the source category, but the morphism map is still from the source to the target category.²

The category **DCont** of directed containers is equivalent to the opposite category of the category **Cat** of small categories and cofunctors. Given a directed container morphism (t, q) , the corresponding cofunctor is (t, \bar{q}) where \bar{q} is defined by $\bar{q}(s, (t s, p)) = (s, q_s p)$.

Container functors with a monad structure can be also characterized in terms of additional structure on containers. This structure, studied by us [17] under the name of mnd-containers, is very different from directed containers. Mnd-containers can be seen as a version of nonsymmetric operads where operations may have infinite arities, arguments places of operations are identified nominally rather than positionally and arguments may be discarded and duplicated in composition.

3 Decomposing directed container morphisms

We now show that every morphism between two (directed) containers admits a natural factorization through a third (directed) container, an idea we promote to general functors and comonads in the next section.

It is almost immediate that every container morphism between two containers factorizes through a container with the shapes of the first and positions of the second container.

² A cofunctor looks a bit like a split opcleavage, but is not one. Before we learned about Aguiar's terminology, we spoke of a "relative split pre-opcleavage". See [6] for more discussion on this matter.

14:6 Decomposing Comonad Morphisms

► **Proposition 1.** *Given two containers $C = (S, P)$, $C' = (S', P')$, a morphism $h = (t, q)$ between them factorizes through a third container C^* as below*

$$\begin{array}{ccc} & & h \\ & \curvearrowright & \\ C & \xrightarrow{h^1} & C^* \xrightarrow{h^2} C' \end{array}$$

with the properties that

- $h^1 : C \rightarrow C^*$ is identity on shapes and
- $h^2 : C^* \rightarrow C'$ is identity on positions in every shape.

Proof. We define $C^* = (S^*, P^*)$ where $S^* = S$, $P^* s = P'(t s)$. I.e., C^* has the shapes of the first, but positions of the second container. The corresponding container morphisms are defined as $h^1 = (t^1, q^1)$ and $h^2 = (t^2, q^2)$ where $t^1 s = s$, $q_s^1 p = q_s p$, $t^2 s = t s$, $q_s^2 p = p$. ◀

At the level of functors and natural transformations, this is to say that a natural transformation $(\Sigma s : S.P s \Rightarrow -) \rightarrow (\Sigma s : S'.P' s \Rightarrow -)$, which we know must always be of the form $\lambda(s, v). (t s, v \circ q_s)$, always factors through the functor $\Sigma s : S.P'(t s) \Rightarrow -$.

Considerably more interestingly, this proposition can be strengthened to a factorization of any directed container morphism, in other words, of any morphism between two container comonads.

► **Proposition 2.** *If, in the situation of Proposition 1, C and C' come with directed container structures $(\downarrow, \circ, \oplus)$ resp. $(\downarrow', \circ', \oplus')$, and h is a directed container morphism, then C^* also carries a directed container structure, and h^1, h^2 are directed container morphisms.*

Proof. We define the directed container structure on C^* as $s \downarrow^* p = s \downarrow q_s p$, $\circ_s^* = \circ'_{t s}$, $p \oplus_s^* p' = p \oplus'_{t s} p'$. It is straightforward to verify that these data obey the laws a directed container:

$$\begin{aligned} s \downarrow^* \circ_s^* &= s \downarrow q_s \circ'_{t s} = s \downarrow \circ_s = s \\ s \downarrow^* (p \oplus_s^* p') &= s \downarrow q_s (p \oplus'_{t s} p') = s \downarrow (q_s p \oplus_s q_s \downarrow q_s p p') \\ &= (s \downarrow q_s p) \downarrow q_s \downarrow q_s p p' = (s \downarrow^* p) \downarrow q_s \downarrow^* p p' = (s \downarrow^* p) \downarrow^* p' \\ \circ_s^* \oplus_s^* p &= \circ'_{t s} \oplus'_{t s} p = p \\ p \oplus_s^* \circ_{s \downarrow^* p}^* &= p \oplus'_{t s} \circ'_{t(s \downarrow^* p)} = p \oplus'_{t s} \circ'_{t(s \downarrow q_s p)} = p \oplus'_{t s} \circ'_{t s \downarrow p} = p \\ (p \oplus_s^* p') \oplus_s^* p'' &= (p \oplus'_{t s} p') \oplus'_{t s} p'' = p \oplus'_{t s} (p' \oplus'_{t s \downarrow p} p'') \\ &= p \oplus'_{t s} (p' \oplus'_{t(s \downarrow q_s p)} p'') = p \oplus_s^* (p' \oplus_s^* \downarrow q_s p p'') = p \oplus_s^* (p' \oplus_s^* \downarrow^* p p'') \end{aligned}$$

That h^1 and h^2 satisfy the directed container morphism laws is also straightforward. ◀

Let us now see what this means on our examples of directed container morphisms.

- Let $(S, P, \downarrow, \circ, \oplus)$ and $(S', P', \downarrow', \circ', \oplus')$ be the directed containers for the nonempty lists comonad and the streams comonad respectively. Recall that we considered the directed container morphism given by $t n = *$ and $q_n i = \min(i, n)$. This directed container morphism factors through the directed container $(S^*, P^*, \downarrow^*, \circ^*, \oplus^*)$ defined by $S^* = \mathbb{N}$, $P^* n = \mathbb{N}$, $n \downarrow^* i = n - \min(i, n)$, $\circ_n^* = 0$, $i \oplus_n^* j = i + j$. This corresponds to the comonad defined by $D^* X = \mathbb{N} \times X^\omega \cong \Sigma n : \mathbb{N}. \mathbb{N} \Rightarrow X$, $\varepsilon^*(n, (x_0, x_1, \dots)) = x_0$, $\delta^*(n, (x_0, x_1, \dots)) = (n, ((n, (x_0, x_1, \dots)), (n-1, (x_1, x_2, \dots)), \dots, (0, (x_n, x_{n+1}, \dots))))$,

$(0, (x_{n+1}, x_{n+2}, \dots), \dots)$). It may be helpful to think of elements of this type as streams with a trusted initial segment: in the datastructure $(n, (x_0, x_1, \dots))$, the elements (x_0, x_1, \dots, x_n) are trusted.

- Let both $(S, P, \downarrow, \circ, \oplus)$ and $(S', P', \downarrow', \circ', \oplus')$ be the directed container for the nonempty lists comonad. Let us consider $t n = n \div 2$ and $q_n i = 2 * i$. This directed container morphism factors through the directed container $(S^*, P^*, \downarrow^*, \circ^*, \oplus^*)$ defined by $S^* = \mathbb{N}$, $P^* n = [0..n \div 2]$, $n \downarrow^* i = n - 2 * i$, $\circ_n^* = 0$, $i \oplus_n^* j = i + j$. This corresponds to the comonad defined by $D^* X = 2 \times X^+ \cong \Sigma n : \mathbb{N}. [0..n \div 2] \Rightarrow X$, $\varepsilon^*(b, (x_0, x_1, \dots, x_m)) = x_0$, $\delta^*(b, (x_0, x_1, \dots, x_m)) = (b, (b, (x_0, x_1, \dots, x_m)), (b, (x_1, \dots, x_m)), \dots, (b, (x_m)))$. Here the thinking is that to recover n from $m = n \div 2$, one has to also know the parity b of n .
- Let $(S, P, \downarrow, \circ, \oplus)$ and $(S', P', \downarrow', \circ', \oplus')$ be the directed containers for the nonempty lists comonad and the labelled finite binary trees comonad. Let $t n = \{w \in 2^* \mid |w| = n\}$ and $q_n u = |u|$. This directed container morphism factors through the directed container $(S^*, P^*, \downarrow^*, \circ^*, \oplus^*)$ defined by $S^* = \mathbb{N}$, $P^* n = \{u \in 2^* \mid |u| \leq n\}$, $n \downarrow^* u = n - |u|$, $\circ_n^* = ()$ and $u \oplus_n^* v = u \cdot v$. This is the comonad of labelled perfectly balanced binary trees.
- Let $(S, P, \downarrow, \circ, \oplus)$ and $(S', P', \downarrow', \circ', \oplus')$ be the directed containers for the labelled finite binary trees comonad and nonempty lists comonad. Let $t b$ be the length of the unique prefix in b of the stream 0^ω , i.e., the unique n such that $0^n \in b$. Let $q_b i = 0^i$. The directed container morphism (t, q) factors through the directed container $(S^*, P^*, \downarrow^*, \circ^*, \oplus^*)$ defined by $S^* = \text{“bars”}$, $P^* b = [0..t b]$, $b \downarrow^* i = \{v \mid 0^i \cdot v \in b\}$, $\circ_b^* = 0$, $i \oplus_b^* j = i + j$. This is a comonad of finite binary trees labelled along the leftmost path only. The counit extracts the label of the root node of the given tree. The comultiplication replaces the label of each node on the leftmost path with the subtree rooted by that node.

That the above-described factorization of container comonad morphisms should be possible is curious and by no means “granted”. Strengthening the factorization of Proposition 1 to morphisms between container monads, for instance, does not work: the middle container functor is generally not a monad and the two natural transformations are not monad morphisms. Indeed, should C, C' be mnd-containers in the sense of [17], C^* will in general not be a mnd-container. For it to be one, from the given operations $\bullet : (\Sigma s : S. P s \Rightarrow S) \rightarrow S$ and $\bullet' : (\Sigma s : S'. P' s \Rightarrow S') \rightarrow S'$ (the shape maps for the multiplications of the corresponding monads), we would need to produce an operation $\bullet^* : (\Sigma s : S. P'(t s) \Rightarrow S) \rightarrow S$ (the shape map for the multiplication of a hypothetical middle monad). To define such an operation \bullet^* in terms of \bullet , we would need a way to turn a given function $v : P'(t s) \rightarrow S$ into a function $v' : P s \rightarrow S$, but we cannot, since we cannot invert $q_s : P'(t s) \rightarrow P s$. To define \bullet^* in terms of \bullet' we would need to be able to convert a given shape $s : S'$ into a shape $s' : S$, but we cannot invert $t : S \rightarrow S'$ in general.

Let us also note that, in the light of the equivalence of \mathbf{DCont} and $(\overleftarrow{\mathbf{Cat}})^{\text{op}}$, our factorization of directed container morphisms is reminiscent of the full image factorization of functors [15]. In fact, it was the full image factorization that first lead us to the above factorization of directed container morphisms. Specifically, given a functor $F : \mathcal{C} \rightarrow \mathcal{D}$, its *full image* is the category $\overline{\text{im}} F$ with as objects the objects of \mathcal{C} , and as morphisms $X \rightarrow Y$ the morphisms $F X \rightarrow F Y$ of \mathcal{D} . The full image of F also comes with two functors: $\overline{F} : \mathcal{C} \rightarrow \overline{\text{im}} F$ that acts as identity on objects and as F on morphisms, and $\overline{\overline{F}} : \overline{\text{im}} F \rightarrow \mathcal{D}$ that acts as F on objects and as identity on morphisms. As such, \overline{F} is the analogue of h^1 and $\overline{\overline{F}}$ the analogue of h^2 in the factorization of a directed container morphism h , as defined above. In the next section, we will see that we are indeed dealing with an analogue of full image factorization for cofunctors.

14:8 Decomposing Comonad Morphisms

Given a container $C' = (S', P')$, a coalgebra structure with carrier S of $\llbracket C' \rrbracket^c$ is a map $\gamma : S \rightarrow \Sigma s : S'. P' s \Rightarrow S$, which splits into $t : S \rightarrow S'$ and $q : \Pi_{s:S}. P' (t s) \rightarrow S$. These are exactly the data of a container morphism from the costate container for S to C' . If $E' = (S', P', \downarrow', \circ', \oplus')$ is a directed container, then γ is a coalgebra of the container comonad $\llbracket E' \rrbracket^{dc}$ iff t, q satisfy $t (q_s p) = t s \downarrow' p$, $s = q_s \circ'_{ts}$, $q_{q_s p} p' = q_s (p \oplus'_{ts} p')$. These laws coincide with those of a directed container morphism from the costate directed container for S to E' .

Hence the factorization of morphisms between container functors (comonads) immediately gives us a factorization of container functor (comonad) coalgebra structures: a functor (comonad) coalgebra structure $\gamma : S \rightarrow \Sigma s : S'. P' s \Rightarrow S$ given by (t, q) factors as a composition of a functor (comonad) coalgebra structure $\gamma^* : S \rightarrow \Sigma s : S. P' (t s) \Rightarrow S$ given by (id_S, q) and a natural transformation (comonad morphism) given by $(t, \lambda_s \cdot \text{id}_{P'(ts)})$.

4 Decomposing general comonad morphisms

We now proceed to showing that the observations we made about morphisms between container comonads on **Set** hold about general comonads on general categories, under some assumptions. Specifically, they hold for comonad morphisms to comonads whose underlying functor preserves Cartesian natural transformations to itself on any category \mathcal{C} with finite limits. For this, we first need to generalize identity-on-shapes and identity-on-positions-in-every-shape directed container morphisms to general comonad morphisms.

For an endofunctor D , which we think of as a datatype, we proceed from the idea that the shape of a datastructure in DX is its image under $D!_X$ in $D1$, which we treat as the object of shapes of D . A natural transformation $\phi : D \rightarrow D'$ between two datatypes can thus be considered bijective as a shape map if $\phi_1 : D1 \rightarrow D'1$ is an isomorphism.

We avoid introducing any objects of positions. We just think of a natural transformation $\psi : D^* \rightarrow D'$ as bijective as a position map for any shape in D^*1 and its image under ψ_1 in $D'1$ if ψ is Cartesian, i.e., if all its naturality squares

$$\begin{array}{ccc} D^* X & \xrightarrow{\psi_X} & D' X \\ D^* f \downarrow & & \downarrow D' f \\ D^* Y & \xrightarrow{\psi_Y} & D' Y \end{array}$$

are pullbacks. This is motivated by the following considerations. In the presence of a terminal object 1 , it is sufficient (while trivially necessary) for Cartesianity of ψ that just the naturality squares for maps $!_X : X \rightarrow 1$, i.e.,

$$\begin{array}{ccc} D^* X & \xrightarrow{\psi_X} & D' X \\ D^* !_X \downarrow & & \downarrow D' !_X \\ D^* 1 & \xrightarrow{\psi_1} & D' 1 \end{array}$$

are pullbacks (cf. [14, Sec. 3.2]), because then, for any $f : X \rightarrow Y$, both the bottom square and outer square in the following diagram are pullbacks, and hence so is the top square, which is the naturality square for f :

$$\begin{array}{ccccc} D^* X & \xrightarrow{\psi_X} & D' X & & \\ & \searrow D^* f & & \searrow D' f & \\ & & D^* Y & \xrightarrow{\psi_Y} & D' Y \\ & \searrow D^* !_X & \downarrow D^* !_Y & & \downarrow D' !_Y \\ & & D^* 1 & \xrightarrow{\psi_1} & D' 1 \end{array}$$

In the case of $\mathcal{C} = \mathbf{Set}$, the naturality square for $!_X$ being a pullback means that D^*X is isomorphic to the set of pairs $(s, xs) : D^*1 \times D'X$ such that $\psi_1 s = D'!_X xs$, i.e., a shape s for D^* together with a datastructure xs in $D'X$ whose shape is the image of s under the shape map ψ_1 . Since the map ψ_X is, up to this isomorphism, just the 2nd projection, and it is also natural in X , it must send datastructures in D^*X to datastructures in $D'X$ linearly, i.e., without discarding or duplicating any data (elements of X) contained in them.

We need to work with endofunctors preserving Cartesian natural transformations to themselves. We say that an endofunctor D' preserves Cartesian natural transformations to D' ³ if, for any endofunctor D and Cartesian natural transformation $\tau : D \rightarrow D'$, the natural transformation with components $D'\tau_X : D'DX \rightarrow D'D'X$ is also Cartesian. This may sound like a peculiar concept but was also needed by Kelly in his work on clubs and datatypes [14, Prop. 3.1]. Container functors have this property since they preserve arbitrary pullbacks.

We first show that natural transformations factorize as expected in the above sense.

► **Theorem 3** (cf. [14, Sec. 3.2]). *Given a category \mathcal{C} with finite limits and two endofunctors D and D' , a natural transformation τ from D to D' admits a factoring through a third endofunctor D^* , as depicted here,*

$$\begin{array}{ccccc} & & \tau_X & & \\ & \curvearrowright & & \curvearrowleft & \\ DX & \xrightarrow{\phi_X} & D^*X & \xrightarrow{\psi_X} & D'X \end{array}$$

with the properties that

- $\phi_1 : D1 \rightarrow D^*1$ is an isomorphism and
- $\psi : D^* \rightarrow D'$ is Cartesian.

Proof. For any X , we construct D^*X together with $\psi_X : D^*X \rightarrow D'X$ and $\pi_X : D^*X \rightarrow D1$ as a pullback. Further, we construct $\phi_X : DX \rightarrow D^*X$ as a unique map to this pullback.

$$\begin{array}{ccccc} DX & \xrightarrow{\tau_X} & D'X & & \\ \phi_X \searrow & & \downarrow D'!_X & & \\ \boxed{B} & & D'1 & & \\ \downarrow \pi_X & & \tau_1 & & \\ \boxed{C} & \xrightarrow{\psi_X} & D^*X & \xrightarrow{\psi_X} & D'X \\ \downarrow \pi_X & & \downarrow \pi_X & & \downarrow D'!_X \\ \boxed{A} & & D1 & \xrightarrow{\tau_1} & D'1 \end{array}$$

The latter construction presupposes commutation of the outer square above, which is immediate by the naturality of τ . Note that \boxed{B} gives us the desired factorization of τ .

For any $f : X \rightarrow Y$, we construct the map $D^*f : D^*X \rightarrow D^*Y$ as a unique map to the pullback D^*Y :

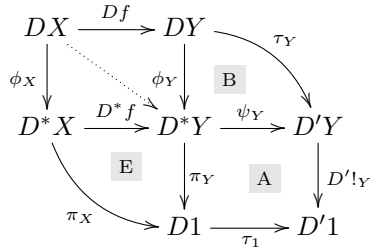
$$\begin{array}{ccccc} D^*X & \xrightarrow{\psi_X} & D'X & & \\ D^*f \searrow & & \downarrow D'f & & \\ \boxed{D} & & D'Y & & \\ \downarrow \pi_X & & \downarrow \pi_Y & & \downarrow D'!_Y \\ \boxed{E} & \xrightarrow{\psi_Y} & D^*Y & \xrightarrow{\psi_Y} & D'Y \\ \downarrow \pi_X & & \downarrow \pi_Y & & \downarrow D'!_Y \\ \boxed{A} & & D1 & \xrightarrow{\tau_1} & D'1 \end{array}$$

³ More precisely, composition with D' from the left preserves them. The terminology is from Garner [9].

14:10 Decomposing Comonad Morphisms

This presupposes the commutativity of the outer square, which follows straightforwardly from **A** and uniqueness of maps to 1. We omit the identity and composition preservation proofs – these follow straightforwardly from id_{D^*X} and $D^*g \circ D^*f$ satisfying the same unique map properties as $D^*\text{id}_X$ and $D^*(g \circ f)$.

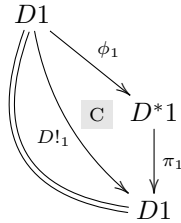
The naturality square of ϕ for a map $f : X \rightarrow Y$ follows from both paths in it satisfying the properties of the unique map to the pullback D^*Y in the diagram



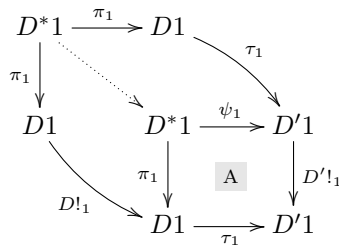
The commutativity of the outer square above follows from **C**, the naturality of τ , and uniqueness of maps to 1.

The naturality of ψ and π are just **D** and **E**.

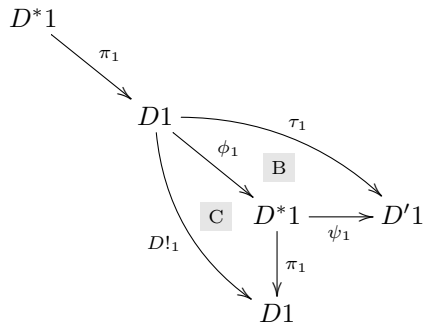
To show that $\phi_1 : D1 \rightarrow D^*1$ is an isomorphism, we prove $\pi_1 : D^*1 \rightarrow D1$ to be its inverse. That the equation $\pi_1 \circ \phi_1 = \text{id}_{D1}$ holds is proved as follows:



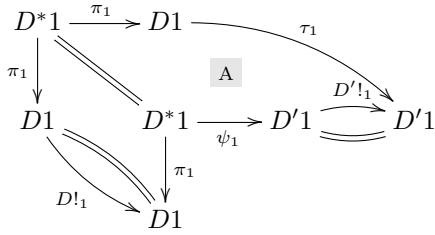
The equation $\phi_1 \circ \pi_1 = \text{id}_{D^*1}$ holds because both sides satisfy the properties of the unique map to the pullback D^*1 in the diagram



where the outer square commutes because $D'^*_1 = \text{id}_{D1}$ and $D'^*_1 = \text{id}_{D^*1}$. Indeed, $\phi_1 \circ \pi_1$ makes the two triangles above commute as follows:



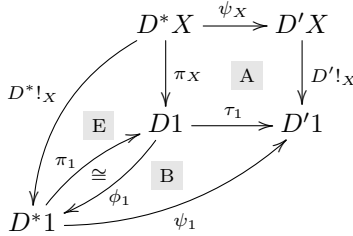
And so does id_{D^*1} :



Finally, we must show that ψ is Cartesian, i.e., that the naturality squares

$$\begin{array}{ccc} D^*X & \xrightarrow{\psi_X} & D'X \\ D^*!_X \downarrow & & \downarrow D'!_X \\ D^*1 & \xrightarrow{\psi_1} & D'1 \end{array}$$

for $!_X : X \rightarrow 1$ are pullbacks. This follows from D^*X being a pullback if we replace the node $D1$ by D^*1 , which we know to be isomorphic:



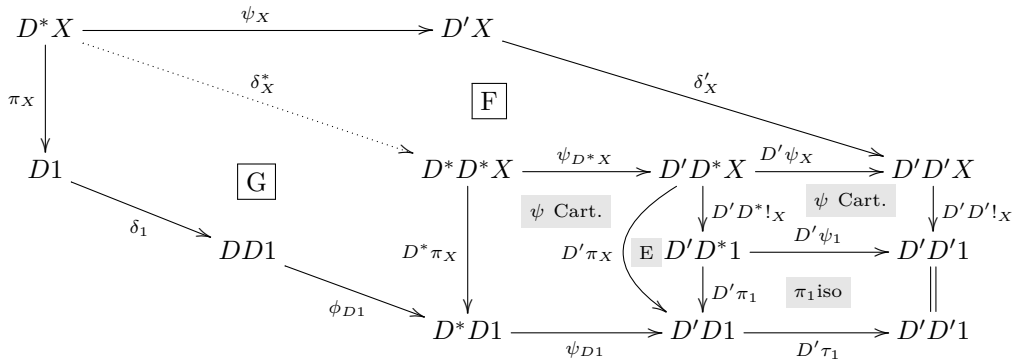
Next we establish that not only do natural transformations factorize, but comonad morphisms do as well.

► **Theorem 4.** *If, in the situation of Theorem 3, D' preserves Cartesian natural transformations to D' , both D and D' carry a comonad structure, and τ is a comonad morphism, then the constructed functor D^* also carries a comonad structure, and ϕ and ψ are comonad morphisms.*

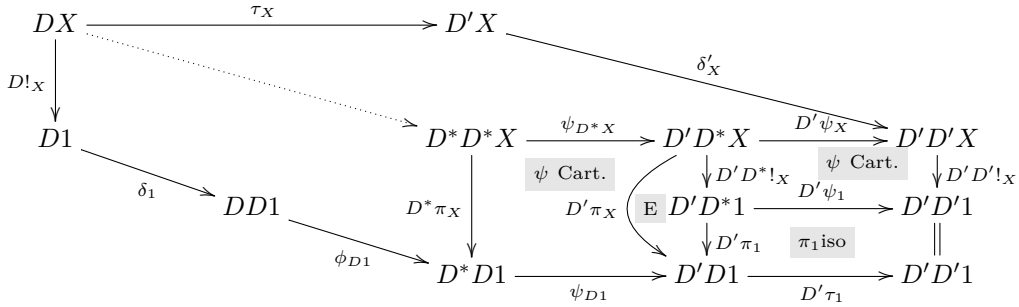
Proof. We define the counit ε^* straightforwardly by

$$\varepsilon_X^* = D^*X \xrightarrow{\psi_X} D'X \xrightarrow{\varepsilon'_X} X$$

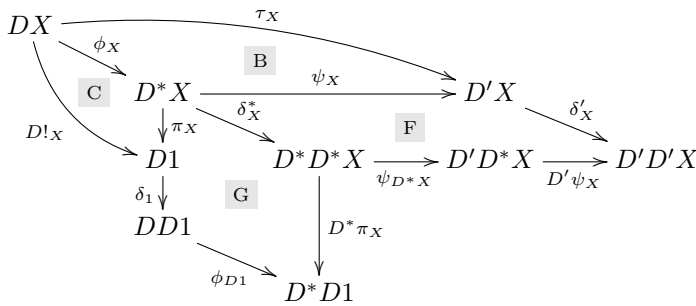
We construct the comultiplication δ^* as a unique map to D^*D^*X as a pullback obtained by pasting three pullbacks (of those, the right upper one is a pullback because D' preserves Cartesian natural transformations to D'):



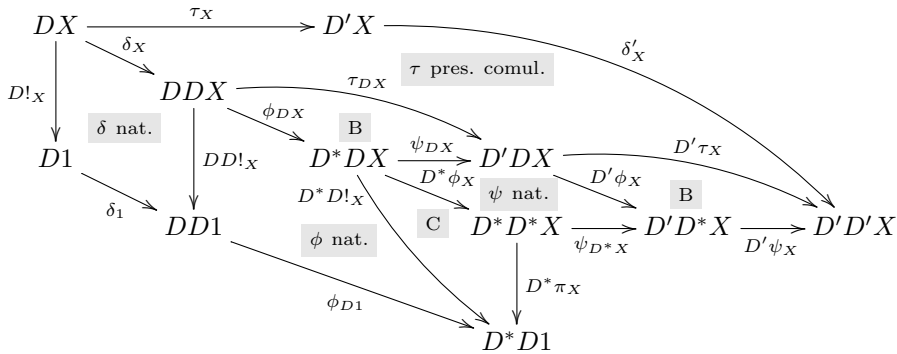
the left-hand and right-hand sides satisfy the properties of a unique map to D^*D^*X :



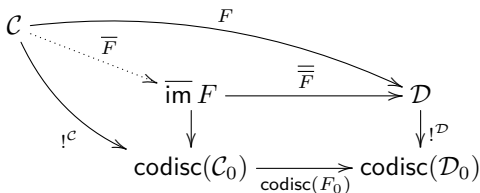
That $\delta'_X \circ \phi_X$ satisfies the two triangles in the above diagram is verified as follows:



That $D^*\phi_X \circ \phi_{DX} \circ \delta_X$ also satisfies the same two triangles is checked as follows:



Let us briefly compare the situation of Theorem 4 with the full image factorization of functors discussed in Section 3. Given a functor $F : \mathcal{C} \rightarrow \mathcal{D}$, the category $\overline{\text{im}} F$, together with the associated functors \overline{F} and \overline{F} , arises as in the following pullback diagram in **Cat**:



where $\text{codisc}(\mathcal{C}_0)$ is the codiscrete category on the set of objects of \mathcal{C} (the cofree category). The arrows $!^{\mathcal{C}}$ and $!^{\mathcal{D}}$ are the unique identity-on-objects functors.

14:16 Decomposing Comonad Morphisms

We are dealing with the following pullback diagram in $\mathbf{Comonad}(\mathcal{C})$:

$$\begin{array}{ccccc}
 D & & \xrightarrow{\tau} & & D' \\
 \downarrow \phi & & & & \downarrow \psi \\
 D & \xrightarrow{\phi} & D^* & \xrightarrow{\psi} & D' \\
 \downarrow \langle D^!, \varepsilon \rangle & & \downarrow & & \downarrow \langle D'^!, \varepsilon' \rangle \\
 D1 \times - & \xrightarrow{\tau_1 \times -} & D'1 \times - & &
 \end{array}$$

This is obtained from the first diagram in the proof of Theorem 3, read as a diagram in $[\mathcal{C}, \mathcal{C}]$ rather than \mathcal{C} , by replacing the constant functors $D1$ and $D'1$ by the corresponding cofree comonads (the coreader comonads for $D1$ and $D'1$). The special case for container comonads is, in the view of the equivalence of \mathbf{DCont} and $(\overleftarrow{\mathbf{Cat}})^{\text{op}}$, an analogue of full image factorization for cofunctors: a pushout diagram in $\overleftarrow{\mathbf{Cat}}$ involving discrete categories.

We do not prove it here, but the factorization asserted in Theorem 3 is unique up to a unique natural isomorphism (cf. [14, Sec. 3.2]). The factorization of Theorem 4 is unique up to a unique isomorphism of comonads. Thus in fact we have factorization systems on $[\mathcal{C}, \mathcal{C}]$ and on the full subcategory of $\mathbf{Comonad}(\mathcal{C})$ given by underlying functors preserving Cartesian natural transformations to themselves. The “epis” of these factorization systems are natural transformations resp. comonad morphisms ϕ such that ϕ_1 is an isomorphism; the “monos” are Cartesian natural transformations resp. comonad morphisms.

We conclude by specializing the above results to the factorization of functor coalgebras and comonad coalgebras. This uses the costate functor and costate comonad.

► **Proposition 5.** *In a Cartesian closed category \mathcal{C} , given an object S , the functor $D^S = S \times (S \Rightarrow -)$ (the costate functor for S) carries a comonad structure (the costate comonad).*

Proof. Immediate from the fact that D^S is defined as the composition of the adjoint functors $S \times -$ and $S \Rightarrow -$. Accordingly, the counit and comultiplication ε^S and δ^S are constructed from the counit and unit of the adjunction: $\varepsilon_X^S = \text{ev}_{S,X} : S \times (S \Rightarrow X) \rightarrow X$, $\delta_X^S = S \times \text{coev}_{S,S \Rightarrow X} : S \times (S \Rightarrow X) \rightarrow S \times (S \Rightarrow (S \times (S \Rightarrow X)))$. ◀

Coalgebras of functors (resp. comonads) are the same as natural transformations (resp. comonad morphisms) from the costate functor (resp. comonad). This result is analogous to the well-known result about algebras of functors (resp. monads) and natural transformations (resp. monad morphisms) to the continuation functor (resp. monad) [13, 10].

► **Proposition 6.**

1. *In a Cartesian closed category \mathcal{C} , given a strong functor D' , there is a bijection between maps from S to $D'S$ and natural transformations from D^S to D' .*
2. *If D' is a comonad, the same bijection restricts to a bijection between comonad coalgebras of D' with carrier S and comonad morphisms from D^S to D' .*

Proof (sketch). We use that tensorially strong functors are internally functorial. We construct the bijection as follows.

Given a map $\gamma : S \rightarrow D'S$, we define a natural transformation $\tau : D^S \rightarrow D'$ by

$$\tau_X = S \times (S \Rightarrow X) \xrightarrow{\gamma \times \text{ifunc}_{S,X}^{D'}} D'S \times (D'S \Rightarrow D'X) \xrightarrow{\text{ev}_{D'S, D'X}} D'X$$

If D' is a comonad and γ satisfies the laws of a comonad coalgebra structure, then τ satisfies the laws of a comonad morphism.

Given a natural transformation $\tau : D^S \rightarrow D'$, we define a map $\gamma : S \rightarrow D'S$ by

$$\gamma = S \xrightarrow{\langle \text{id}_S, !_S \rangle} S \times 1 \xrightarrow{S \times \text{id}_S} S \times (S \Rightarrow S) \xrightarrow{\tau_S} D'S$$

If D' is a comonad and τ satisfies the laws of a comonad morphism, then γ satisfies the laws of a comonad coalgebra structure.

The two transformations are mutual inverses. ◀

Using what we have learned about the costate functor and costate comonad, we obtain a decomposition of functor coalgebras and comonad coalgebras.

► **Theorem 7.**

1. Given a Cartesian closed finitely complete category \mathcal{C} , a strong functor D' preserving Cartesian natural transformations to D' , and a map $\gamma : S \rightarrow D'S$, then γ admits a factoring through the object D^*S for another functor D^* , as depicted below

$$\begin{array}{ccccc} & & \gamma & & \\ & \curvearrowright & & \curvearrowleft & \\ S & \xrightarrow{\gamma^*} & D^*S & \xrightarrow{\psi_S} & D'S \end{array}$$

with the properties that

- $D^*! \circ \gamma^* : S \rightarrow D^*1$ is an isomorphism and
 - $\psi : D^* \rightarrow D'$ is Cartesian.
2. If D' is a comonad and γ is a comonad coalgebra structure, then D^* is a comonad, γ^* is a comonad coalgebra structure and ψ is a comonad morphism.

Proof (sketch). This is a corollary of Theorems 3, 4 and the last two propositions.

The given map $\gamma : S \rightarrow D'S$ induces a natural transformation $\tau : D^S \rightarrow D'$. From this, we get a functor D^* and two natural transformations $\phi : D^S \rightarrow D^*$ and $\psi : D^* \rightarrow D'$, whereby $\phi_1 : D^S1 \rightarrow D^*1$ is an isomorphism and ψ is Cartesian. We construct $\gamma^* : S \rightarrow D^*S$ as the composition $\phi_S \circ (S \times \text{id}_S) \circ \langle \text{id}_S, !_S \rangle$. The map $D^*! \circ \gamma^*$ is an isomorphism thanks to commutation of the diagram

$$\begin{array}{ccccccc} & & & \gamma^* & & & \\ & & & \curvearrowright & & & \\ S & \xrightarrow{\langle \text{id}_S, !_S \rangle} & S \times 1 & \xrightarrow{S \times \text{id}_S} & S \times (S \Rightarrow S) & \xrightarrow{\phi_S} & D^*S \\ & \cong \searrow & & \cong \searrow & \downarrow D^S! & & \downarrow D^*! \\ & & & & S \times (S \Rightarrow 1) & \xrightarrow{\phi_1} & D^*1 \\ & & & & \cong \searrow & & \cong \searrow \end{array}$$

5 Conclusion

We have demonstrated that two observations about comonads that are immediate for container comonads on **Set** also hold more generally for comonads whose underlying functor preserves Cartesian natural transformations to itself on any finitely complete category. These observations concern shapes and positions (in terms of comonad morphisms being bijective on shapes or bijective on positions between corresponding pairs of shapes), and demonstrate that comonads generally, not just container comonads, are usefully analyzed in terms of shapes and positions and exhibit noteworthy properties expressible in these terms.

In other work [6], we have shown that container comonad coalgebras and container comonad morphisms can be seen as generalized asymmetric (i.e., server-client) lenses, which are a device for keeping a client's view of a database in synch with the master copy at a server. Shapes in the two directed containers are states of the two databases, positions are updates. The factorization results presented in this paper say that such lenses factorize into two lenses, whereof the first is identity on states and the second is identity on updates for every state.

References

- 1 Michael Abbott, Thorsten Altenkirch, and Neil Ghani. Containers: Constructing Strictly Positive Types. *Theor. Comput. Sci.*, 342(1):3–27, 2005. doi:10.1016/j.tcs.2005.06.002.
- 2 Marcelo Aguiar. *Internal Categories and Quantum Groups*. PhD thesis, Cornell University, 1997. URL: <http://www.math.cornell.edu/~maguiar/thesis2.pdf>.
- 3 Danel Ahman, James Chapman, and Tarmo Uustalu. When Is a Container a Comonad? *Log. Methods Comput. Sci.*, 10(3), 2014. article 14. doi:10.2168/lmcs-10(3:14)2014.
- 4 Danel Ahman and Tarmo Uustalu. Distributive Laws of Directed Containers. *Progress in Informatics*, 10:3–18, 2013. doi:10.2201/niipi.2013.10.2.
- 5 Danel Ahman and Tarmo Uustalu. Directed Containers as Categories. In Robert Atkey and Neil Krishnaswami, editors, *Proc. of 6th Wksh. on Mathematically Structured Functional Programming, MSFP 2016 (Eindhoven, Apr. 2016)*, volume 207 of *Electron. Proc. in Theor. Comput. Sci.*, pages 89–98. Open Publishing Assoc., 2016. doi:10.4204/eptcs.207.5.
- 6 Danel Ahman and Tarmo Uustalu. Taking Updates Seriously. In Romina Eramo and Michael Johnson, editors, *Proc. of 6th Wksh. on Mathematically Structured Functional Programming, MSFP 2016 (Eindhoven, Apr. 2016)*, volume 1827 of *CEUR Wksh. Proc.*, pages 59–73. RWTH Aachen, 2017. URL: <http://ceur-ws.org/Vol-1827/paper11.pdf>.
- 7 Nicola Gambino and Martin Hyland. Wellfounded Trees and Dependent Polynomial Functors. In Stefano Berardi, Mario Coppo, and Ferruccio Damiani, editors, *Revised Selected Papers from Int. Wksh. on Types for Proofs and Programs, TYPES 2003 (Torino, Apr./May 2003)*, volume 3085 of *Lect. Notes in Comput. Sci.*, pages 210–225. Springer, 2004. doi:10.1007/978-3-540-24849-1_14.
- 8 Nicola Gambino and Joachim Kock. Polynomial Functors and Polynomial Monads. *Math. Proc. Cambridge Philos. Soc.*, 154(1):153–192, 2013. doi:10.1017/s0305004112000394.
- 9 Richard Garner. Double Clubs. *Cahiers de Topologie et Géométrie Différentielle Catégoriques*, 47(4):261–316, 2006. URL: http://www.numdam.org/item?id=CTGDC_2006__47_4_261_0.
- 10 Martin Hyland, Paul B. Levy, Gordon Plotkin, and John Power. Combining Algebraic Effects with Continuations. *Theor. Comput. Sci.*, 375(1–3):20–40, 2007. doi:10.1016/j.tcs.2006.12.026.
- 11 C. Barry Jay. A Semantics for Shape. *Sci. Comput. Program.*, 25(2–3):251–283, 1995. doi:10.1016/0167-6423(95)00015-1.
- 12 C. Barry Jay and J. Robin B. Cockett. Shapely Types and Shape Polymorphism. In Donald Sannella, editor, *Proc. of 5th European Symp. on Programming Languages and Systems, ESOP '94 (Edinburgh, Apr. 1994)*, volume 788 of *Lect. Notes in Comput. Sci.*, pages 302–316. Springer, 2004. doi:10.1007/3-540-57880-3_20.
- 13 G. Max Kelly. A Unified Treatment of Transfinite Constructions for Free Algebras, Free monoids, Colimits, Associated Sheaves, and So on. *Bull. Austral. Math. Soc.*, 22(1):1–83, 1980. doi:10.1017/s0004972700006353.
- 14 G. Max Kelly. On Clubs and Data-Type Constructors. In Michael P. Fourman, Peter T. Johnstone, and Andrew M. Pitts, editors, *Applications of Categories in Computer Science*, volume 177 of *London. Math. Soc. Lect. Note Series*, pages 163–190. Cambridge Univ. Press, 1992. doi:10.1017/cbo9780511525902.010.

- 15 nLab authors. Full Image. nLab entry, revision 5, April 2016. URL: <http://ncatlab.org/nlab/revision/full%20image/5>.
- 16 John Power and Olha Shkaravska. From Comodels to Coalgebras: State and Arrays. *Electron. Notes Theor. Comput. Sci.*, 106:297–314, 2004. doi:10.1016/j.entcs.2004.02.041.
- 17 Tarmo Uustalu. Container Combinatorics: Monads and Lax Monoidal Functors. In Mohammad Reza Mousavi and Jiří Sgall, editors, *Proc. of 2nd IFIP WG 1.8 Int. Conf. on Topics in Theoretical Computer Science, TTCS 2017 (Tehran, Sept. 2017)*, volume 10608 of *Lect. Notes in Comput. Sci.*, pages 91–105. Springer, 2017. doi:10.1007/978-3-319-68953-1_8.
- 18 Mark Weber. Polynomials in Categories with Pullbacks. *Theor. Appl. Categ.*, 30(16):533–598, 2015. URL: <http://www.tac.mta.ca/tac/volumes/30/16/30-16abs.html>.

The Axiom of Choice in Cartesian Bicategories

Filippo Bonchi

University of Pisa, Italy

Jens Seeber

IMT School for Advanced Studies Lucca, Italy

Paweł Sobociński

University of Southampton, UK

Abstract

We argue that cartesian bicategories, often used as a general categorical algebra of relations, are also a natural setting for the study of the axiom of choice (AC). In this setting, AC manifests itself as an inequation asserting that every total relation contains a map. The generality of cartesian bicategories allows us to separate this formulation from other set-theoretically equivalent properties, for instance that epimorphisms split. Moreover, via a classification result, we show that cartesian bicategories satisfying choice tend to be those that arise from bicategories of spans.

2012 ACM Subject Classification Theory of computation → Categorical semantics; Theory of computation → Logic

Keywords and phrases Cartesian bicategories, Axiom of choice, string diagrams

Digital Object Identifier 10.4230/LIPIcs.CALCO.2019.15

Introduction

Cartesian bicategories were introduced by Carboni and Walters [8] as a categorical algebra of relations and an alternative to Freyd and Scedrov’s allegories [14]¹. In recent years they have been receiving renewed attention by researchers interested in string-diagrammatic languages. Indeed, thanks to the compact closed structure induced by Frobenius bimonoids, cartesian bicategories have proved to be an appropriate mathematical playground for compositional studies of different kinds of feedback systems. For instance, signal flow graphs [21], which are circuit-like specifications of linear dynamical systems, form a cartesian bicategory [3]. Moreover, the fact that cartesianity only holds laxly makes them able to serve as “resource-sensitive” syntax, as outlined in [4], where free cartesian bicategories were proposed as a resource-sensitive generalisation of Lawvere theories.

Free cartesian bicategories were also used in [5], where we showed that their algebraic presentation can be seen as an equational characterisation of well-known logical preorders, namely those arising from query inclusion of conjunctive queries (aka regular logic). The deep relationship between cartesian bicategories and regular logic – already alluded to in [8] – was also recently touched upon by Fong and Spivak [11].

In cartesian bicategories, it is important to distinguish between arbitrary morphisms – which can be thought of as relations – and a certain class of morphisms called *maps*, which can be thought of as functions. A fundamental result [8, Theorem 3.5] states that, for a cartesian bicategory \mathcal{B} satisfying the property of *functional completeness*, (i) the subcategory of maps (denoted by $\text{Map } \mathcal{B}$) is regular and (ii) the category of relations over the category of maps ($\text{Rel}(\text{Map } \mathcal{B})$) is biequivalent to \mathcal{B} . Unfortunately, this beautiful result is not relevant for

¹ RFC Walters referred to the modular law of allegories as a *formica mentale*, a “complication which prevents thought” (<http://rfcwalters.blogspot.com/2009/10/categorical-algebras-of-relations.html>).



free cartesian bicategories: for instance the categories obtained by the algebraic presentations in [4] and [5] do not arise from the $\mathbf{Rel}(\cdot)$ construction.

For this reason in [5], we needed to rely on an alternative construction \mathbf{Span}^\sim that we believe is of independent interest. First, it requires less structure of the underlying category: while $\mathbf{Rel}(\cdot)$ requires a regular category, \mathbf{Span}^\sim requires merely the presence of weak pullbacks. Second, while in the category of sets and functions both constructions yield the usual category of relations, as we shall see, there are important cases in which they differ.

Our main contribution is an analogue of the aforementioned result for \mathbf{Span}^\sim , namely that $\mathbf{Span}^\sim \mathbf{Map} \mathcal{B}$ is biequivalent to \mathcal{B} . In this setting, Carboni and Walters’ functional completeness can be relaxed to a weaker condition that we call *having enough maps*, but an additional assumption is necessary: \mathcal{B} has to satisfy the *axiom of choice*. Indeed, our main result (Theorem 30) asserts that a cartesian bicategory \mathcal{B} with enough maps satisfies the axiom of choice *if and only if* \mathcal{B} is biequivalent to $\mathbf{Span}^\sim \mathbf{Map} \mathcal{B}$.

This characterisation motivates a closer look at the axiom of choice, one of the best known – and most controversial – axioms of set theory [16]. It has many ZF-equivalent formulations, some requiring only very basic concepts. One is:

Every total relation contains a map.

Our starting observation is that this condition is natural to state in the language of cartesian bicategories. Another way of viewing our main result is, therefore, a characterisation of cartesian bicategories with enough maps that satisfy the axiom of choice as precisely those that arise via the \mathbf{Span}^\sim construction.

Given the innovations of topos theory [19] in foundations of mathematics, the question of whether or not to accept the axiom of choice is nowadays less absolute (and therefore less heated). Indeed, if a topos is a mathematical “universe”, then it holds in some and not in others, thus accepting/rejecting choice turns from a philosophical question into a practical matter. Interpreting choice inside a category does not need the full power of the internal language of a topos – it suffices if the category in question captures basic properties of relations. Cartesian bicategories can therefore be seen as an amusing setting for the study of the axiom of choice. Indeed, the advantage of a weaker language is a finer grained analysis: e.g. we shall see that properties well-known to be equivalent to choice in ZF (e.g. surjective functions split) are different as properties of cartesian bicategories.

Structure of the paper. We start by giving an overview of a few important concepts of cartesian bicategories in Section 1. In Section 2 we define the axiom of choice in cartesian bicategories, the property of “having enough maps” and discuss ramifications of this, including a useful characterisation. In Section 3 we introduce the \mathbf{Span}^\sim construction and prove several useful results that are necessary for showing the classification theorem in Section 4. In Section 5, we compare the constructions $\mathbf{Rel}(\mathcal{C})$ and $\mathbf{Span}^\sim \mathcal{C}$ and show that they coincide if regular epis split in \mathcal{C} .

We would like to thank Aleks Kissinger and the team behind TikZiT, which was used to create the diagrams in this paper.

1 Cartesian bicategories

We start by recalling the notion of cartesian bicategory [8]. We will often use *string diagrams* [23] as a graphical notation for morphisms: given a symmetric monoidal category \mathcal{B} with monoidal product \otimes and monoidal unit I , a wire $\text{—}x\text{—}$ denotes id_X , the identity for an

arbitrary object X of \mathcal{B} , while a box $X \boxed{R} Y$ denotes an arbitrary morphism $R: X \rightarrow Y$ of \mathcal{B} ; for $R: X \rightarrow Y$ and $S: Y \rightarrow Z$, the composition $R; S: X \rightarrow Z$ is depicted as $X \boxed{R} Y \boxed{S} Z$; for $R: X \rightarrow Y$ and $S: Z \rightarrow W$, $R \otimes S: X \otimes Z \rightarrow Y \otimes W$ is depicted as $X \boxed{R} Y \boxed{S} Z$; symmetries $\sigma: X \otimes Y \rightarrow Y \otimes X$ are drawn as $\begin{matrix} X & & Y \\ & \searrow & / \\ & X & & Y \\ & / & \searrow \\ Y & & X \end{matrix}$; the identity for I as an empty diagram $\boxed{\quad}$. When clear from the context, we will avoid labelling wires.

► **Definition 1.** A cartesian bicategory is a symmetric monoidal category \mathcal{B} enriched over the category of posets. Every object $X \in \mathcal{B}$ is equipped with morphisms

$$\overset{x}{\curvearrowright} : X \rightarrow X \otimes X \quad \text{and} \quad \overset{x}{\bullet} : X \rightarrow I$$

such that

- $\overset{x}{\curvearrowright}$ and $\overset{x}{\bullet}$ form a cocommutative comonoid, that is they satisfy

- $\overset{x}{\curvearrowright}$ and $\overset{x}{\bullet}$ have right-adjoints \curvearrowright^x and \bullet^x respectively, that is

- The Frobenius law holds, that is

- Each morphism $R: X \rightarrow Y$ is a lax comonoid homomorphism, that is

- The choice of comonoid on every object is coherent with the monoidal structure² in the sense that

A morphism of cartesian bicategories is a monoidal functor preserving the ordering and the chosen monoids and comonoids.

² In the original definition of [8] this property is replaced by requiring the uniqueness of the comonoid/-monoid. However, as suggested in [22], coherence seems to be the property of primary interest.

15:4 The Axiom of Choice in Cartesian Bicategories

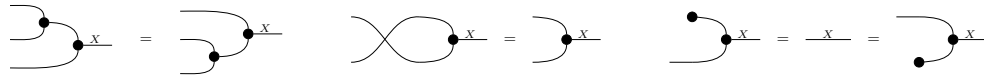
The archetypal example of a cartesian bicategory is the category of sets and relations **Rel**, with cartesian product of sets, hereafter denoted by \times , as monoidal product and $1 = \{\bullet\}$ as unit I . To be precise, **Rel** has sets as objects and relations $R \subseteq X \times Y$ as arrows $X \rightarrow Y$. Composition and monoidal product are defined as expected:

$$R; S = \{(x, z) \mid \exists y \text{ s.t. } (x, y) \in R \text{ and } (y, z) \in S\},$$

$$R \otimes S = \{((x_1, x_2), (y_1, y_2)) \mid (x_1, y_1) \in R \text{ and } (x_2, y_2) \in S\}.$$

For each set X , the comonoid structure is given by the diagonal function $X \rightarrow X \times X$ and the unique function $X \rightarrow 1$, considered as relations. That is $\overset{X}{\curvearrowright} = \{(x, (x, x)) \mid x \in X\}$ and $\overset{X}{\bullet} = \{(x, \bullet) \mid x \in X\}$. Their right adjoints are given by their opposite relations: $\curvearrowright^X = \{((x, x), x) \mid x \in X\}$ and $\bullet^X = \{(\bullet, x) \mid x \in X\}$. The reader can easily check that the four inequalities are satisfied and that, moreover, the Frobenius law holds. The right adjoints also enjoy an additional property that holds in any cartesian bicategory.

► **Lemma 2.** \curvearrowright^X and \bullet^X form a commutative monoid, that is



To appreciate the property that every morphism is a lax-comonoid homomorphism, it is useful to spell out its meaning in **Rel**: in the first inequality, the left and the right-hand side are, respectively, the relations

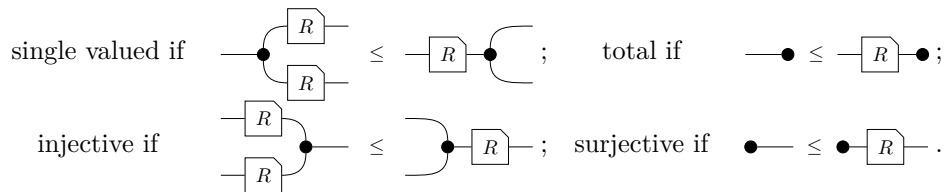
$$\{(x, (y, y)) \mid (x, y) \in R\} \quad \text{and} \quad \{(x, (y, z)) \mid (x, y) \in R \text{ and } (x, z) \in R\}, \tag{1}$$

while in the second inequality, they are the relations

$$\{(x, \bullet) \mid \exists y \in Y \text{ s.t. } (x, y) \in R\} \quad \text{and} \quad \{(x, \bullet) \mid x \in X\}. \tag{2}$$

It is immediate to see that the two left-to-right inclusions hold for any relation $R \subseteq X \times Y$, while the right-to-left inclusions hold exactly when R is a function: a relation which is single valued and total³. This observation justifies the following definition.

► **Definition 3.** Let R be a morphism in a cartesian bicategory. We call R



By translating the last two inequalities in **Rel**, similarly to what we have shown in (1) and (2), the reader can immediately check that these correspond to the usual properties of injectivity and surjectivity for relations. Moreover, since the converses of these inequalities hold in cartesian bicategories, the four inequalities are actually equalities.

³ Requiring every morphism to be a comonoid homomorphism would make \otimes the categorical product [13] and thus the whole category would be cartesian. Ensuring just *lax*-comonoid homomorphism makes \otimes a certain kind of bi-limit, called in [8], bi-product. This fact explains the name cartesian bicategory.

We can characterise all the notions of Definition 3 equivalently in terms of *opposite morphisms* R^{op} which are defined for any morphism R as follows:

$$\overline{R} := \text{diagram of } R^{\text{op}}$$

► **Proposition 4.** *Let R be a morphism in a cartesian bicategory.*

$$\begin{aligned} \overline{R} \overline{R} &\leq \text{---} && \text{iff } R \text{ is single valued.} && \text{---} \leq \overline{R} \overline{R} && \text{iff } R \text{ is total.} \\ \overline{R} \overline{R} &\leq \text{---} && \text{iff } R \text{ is injective.} && \text{---} \leq \overline{R} \overline{R} && \text{iff } R \text{ is surjective.} \end{aligned}$$

In particular, R is surjective iff R^{op} is total and R is injective iff R^{op} is single valued.

Proof. We show the proofs for single valued and total. The proofs for injectivity and surjectivity are analogous. The last statement follows from the others and the fact that $(R^{\text{op}})^{\text{op}} = R$

■ Let R be single valued. Then

$$\overline{R} \overline{R} = \text{diagram} \leq \text{diagram} \leq \text{diagram} = \text{---}$$

Conversely, if $\overline{R} \overline{R} \leq \text{---}$, then by the Frobenius law one gets

$$\text{diagram} = \text{diagram}$$

and from there

$$\text{diagram} \leq \text{diagram} \leq \text{diagram} = \overline{R} \bullet$$

■ Let R be total. Then

$$\overline{R} \overline{R} = \text{diagram} \geq \text{diagram} \geq \text{diagram} = \text{---}$$

Conversely, if $\text{---} \leq \overline{R} \overline{R}$, then

$$\text{---} \bullet \leq \overline{R} \overline{R} \bullet \leq \overline{R} \bullet$$



► **Definition 5.** A map in a cartesian bicategory is a morphism f that is a comonoid homomorphism, i.e. is single valued and total.

We will write \overline{f} to denote a map f and $\overline{\overline{f}}$ for its opposite. Note that we use lower-case letters for maps and upper-case for arbitrary morphisms. Following the analogy with **Rel**, we will often call arbitrary morphisms of a cartesian bicategory relations.

The original treatment of cartesian bicategories in [8] introduces maps as those morphisms that admit a right-adjoint. We show below that this amounts to the same notion.

15:6 The Axiom of Choice in Cartesian Bicategories

► **Proposition 6.** A morphism \boxed{f} is a map if and only if it has a right adjoint – a morphism R such that $\boxed{R}\boxed{f} \leq \text{—}$ and $\text{—} \leq \boxed{f}\boxed{R}$. In that case, necessarily

$$\boxed{R} = \boxed{f}.$$

Proof. If \boxed{f} is a map, then \boxed{f} is a right-adjoint by Proposition 4.

On the other hand, if \boxed{f} has a right-adjoint R , then it is a map since

$$\text{—} \bullet \leq \boxed{f} \bullet \leq \boxed{f}\boxed{R} \bullet \leq \boxed{f} \bullet \leq \text{—} \bullet$$

and

$$\text{—} \bullet \leq \boxed{f}\boxed{R} \bullet \leq \boxed{f} \bullet$$

Therefore, \boxed{f} is indeed a map and $\boxed{R} = \boxed{f}$ by uniqueness of adjoints. ◀

The identity is a map, and maps are easily shown to be closed under composition, so they constitute a category.

► **Definition 7.** Given a cartesian bicategory \mathcal{B} , we define its category of maps, $\text{Map}(\mathcal{B})$ to have the same objects of \mathcal{B} and as morphism the maps of \mathcal{B} .

By the following proposition, the ordering of \mathcal{B} becomes trivial when restricted to maps.

► **Proposition 8.** Let f, g be maps such that $f \leq g$. Then $f = g$.

Proof. Since $\boxed{f} \leq \boxed{g}$, also $\boxed{f} \leq \boxed{g}$. Therefore

$$\boxed{g} \leq \boxed{f}\boxed{f}\boxed{g} \leq \boxed{f}\boxed{g}\boxed{g} \leq \boxed{f} \quad \blacktriangleleft$$

We have seen that **Rel** is source of intuition for cartesian bicategories. There are many other similar examples; for instance **LinRel**, the category of linear relations of vector spaces where the monoidal product is the direct sum of vector spaces. Nevertheless, there are examples of cartesian bicategories that are significantly different, e.g. in which – concretely speaking – the monoidal product does not act as cartesian product on the underlying sets.

► **Example 9.** Recall that a prop is a strict symmetric monoidal category where the objects are the natural numbers and monoidal product on objects is addition. The prop **ERel** of *equivalence relations* [25, 12, 9, 10, 6] (also called the prop of *corelations*) has objects natural numbers, where $n \in \mathbb{N}$ is thought of as the finite set $\{0, \dots, n-1\}$. A morphism $n \rightarrow m$ is an equivalence relation on $n+m$. Composition of an equivalence relation on $n+m$ with one on $m+o$ is given by taking the smallest equivalence relation they generate on $n+m+o$ and restricting it to $n+o$. Monoidal product is given by disjoint union.

Another important example is the prop **PERel** of *partial equivalence relations*. These are symmetric and transitive, but not necessarily reflexive, and have been used in the study of the semantics of higher order λ -calculi [17, 24] and quantum computations [18, 15]. In **PERel** a morphism $n \rightarrow m$ is a partial equivalence relation on $n+m$; composition similar to that in **ERel**, taking the smallest induced partial equivalence relation. Again \otimes is given by disjoint union. See [25, Definitions 2.52 and 2.63] for additional details.

Both **ERel** and **PERel** carry the structure of cartesian bicategories after taking into consideration their posetal enrichment. Here the ordering \leq is the *opposite* of set inclusion: $R \leq S$ iff $R \supseteq S$. Note that for **PERel**, we need some extra care. We consider partial

equivalence relations $R, S: n \rightarrow m$ as equivalence relations \bar{R}, \bar{S} over $(n + m) \cup \{\perp\}$ and then take $R \leq S$ iff $\bar{R} \supseteq \bar{S}$. In particular, notice that the completely undefined partial equivalence relation is represented by the chaotic relation on $(n + m) \cup \{\perp\}$, and is thus – according to this ordering – the least element in its homset.

To define the comonoid structure it is enough to consider 1, since for arbitrary n it is forced by coherence (Definition 1). For both **ERel** and **PERel** $\text{---}\bullet\text{---}$: $1 \rightarrow 2$ is the equivalence relation equating all the elements of the set $1 + 2$ and $\text{---}\bullet\text{---}$: $1 \rightarrow 0$ equates the single element of the set 1. The monoid structure $\text{---}\bullet\text{---}$: $2 \rightarrow 1$ and $\bullet\text{---}$: $0 \rightarrow 1$ is defined in a similar way.

In order to illustrate what maps are in these categories, it is convenient to write $[i]_R$ for the set $\{j \mid (i, j) \in R\}$. Both in **ERel** and **PERel** a morphism $R: n \rightarrow m$ is

$$\text{total iff for all } i, j \in n, (i, j) \in R \text{ implies } i = j, \text{ and} \tag{3}$$

$$\text{single valued iff for all } i \in m, \text{ either } [i]_R = \emptyset \text{ or there is } j \in n \text{ such that } (i, j) \in R. \tag{4}$$

Thus $\text{---}\bullet\text{---}$ is single valued but not total; $\bullet\text{---}$ is total but not single valued. In **PERel**, the undefined relation $0 \rightarrow 1$, hereafter denoted $\boxed{\perp}\text{---}$, is both total and single valued.

2 Choice in Cartesian bicategories

One of the many equivalent formulations of the axiom of choice in set theory is

Every total relation contains a map.

In a total relation every element in the domain is related to at least one element in the codomain. A map is obtained by choosing, for each element in the domain, exactly one related element in the codomain. This can be stated in the language of cartesian bicategories.

► **Definition 10** (Choice). Let \mathcal{B} be a cartesian bicategory. We say that \mathcal{B} satisfies the axiom of choice (AC), or that \mathcal{B} has choice, iff the following holds for any morphism $R: X \rightarrow Y$:

$$\text{---}\bullet \leq \boxed{R}\bullet \text{ (} R \text{ is total)} \text{ implies } \exists \text{ map } f: X \rightarrow Y \text{ such that } \boxed{f} \leq \boxed{R} \text{ (AC)}$$

Observe that the converse implication holds in any cartesian bicategory.

► **Lemma 11.** *If $\boxed{f} \leq \boxed{R}$ then $\text{---}\bullet \leq \boxed{R}\bullet$.*

Proof. Obvious, since if S is total and $S \leq R$, then R is total: $\boxed{R}\bullet \geq \boxed{S}\bullet \geq \text{---}\bullet$. ◀

► **Example 12.**

- The usual axiom of choice implies that **Rel** satisfies (AC).
- **ERel** is an example of a cartesian bicategory that does not satisfy (AC). Recall from Example 9 that the ordering is the *reverse* of inclusion. Therefore, for (AC) to hold would mean that every equivalence relation that satisfies (3) could be included in one that satisfies both (3) and (4). Now consider $\bullet\text{---}$: $0 \rightarrow 1$. As seen in Example 9, it is total, but not single valued. Since equivalence relations have to be reflexive, this is also the only morphism of type $0 \rightarrow 1$: clearly AC fails here.
- Interestingly, **PERel** *does* satisfy (AC). For example, $\bullet\text{---}$: $0 \rightarrow 1$ is included, as an equivalence relation over $(0 + 1) \cup \{\perp\}$, in $\boxed{\perp}\text{---}$.

15:8 The Axiom of Choice in Cartesian Bicategories

Another common formulation of the axiom of choice in set theory is the assertion that every surjective function $\pi: X \rightarrow Y$ splits, namely, there exists a function $\rho: Y \rightarrow X$ such that $\rho; \pi = id_Y$. A standard categorification of the notion of surjectivity is the notion of epi(morphism): π is epi iff $\pi; f = \pi; g$ entails $f = g$. In order to clarify the picture and justify our Definition 10 we will now investigate epimorphisms in cartesian bicategories.

► **Lemma 13.** *Let $\boxed{\pi}$ be a map in a cartesian bicategory \mathcal{B} . Then $\boxed{\pi}$ is an epi in \mathcal{B} if and only if it is surjective.*

Proof. ■ Let π be an epi in \mathcal{B} . Since π is a map, by Proposition 4, $\bullet \leq \boxed{\pi} \boxed{\pi} \bullet$ and therefore

$$\boxed{\pi} \bullet = \bullet = \boxed{\pi} \boxed{\pi} \bullet$$

Since $\boxed{\pi}$ is epi, $\boxed{\pi} \bullet = \bullet$ so $\boxed{\pi}$ is total, hence $\boxed{\pi}$ is surjective by Proposition 4.

■ Assume $\boxed{\pi}$ is surjective. Then $\boxed{\pi} \boxed{\pi} = \bullet$ by Proposition 4. If now R, S are morphisms such that $\boxed{\pi} \boxed{R} = \boxed{\pi} \boxed{S}$, then

$$\boxed{R} = \boxed{\pi} \boxed{\pi} \boxed{R} = \boxed{\pi} \boxed{\pi} \boxed{S} = \boxed{S} \quad \blacktriangleleft$$

► **Lemma 14.** *Surjective maps split in any cartesian bicategory with choice.*

Proof. Let $\pi: X \rightarrow Y$ be a surjective map. Therefore, $\pi^{op}: Y \rightarrow X$ is a total relation, so by (AC) there is a map $g: Y \rightarrow X$ such that

$$\boxed{g} \leq \boxed{\pi}$$

Now we have

$$\boxed{g} \boxed{\pi} \leq \boxed{\pi} \boxed{\pi} \leq \bullet$$

and since both the left hand side and the right hand side of that inequality are maps, we have by Proposition 8 that $g; \pi = id_Y$. ◀

2.1 Cartesian bicategories with enough maps

The converse of Lemma 14 does not hold in general. The reason is that a general cartesian bicategory might not have enough maps to “cover” all its morphisms in a suitable sense. In order to prove the converse, we need to assume a saturation property.

► **Definition 15.** We say a cartesian bicategory *has enough maps* if for every morphism $R: X \rightarrow I$ there is a map $f: Z \rightarrow X$ such that

$$\boxed{R} = \boxed{f} \bullet$$

The intuition for this notion is the following: a morphism $R: X \rightarrow I$ can be considered as a predicate on X . Then having enough maps ensures the existence of a function f that picks out the subset of X where R holds.

► **Example 16.** The description above shows that **Rel** has enough maps. Also **ERel** and **PERel** have both enough maps. We briefly describe the construction for **ERel**, the one for **PERel** is similar. For any morphism $R: n \rightarrow 0$ in **ERel**, take e to be the number of the equivalence classes of R . Choose a total ordering for these equivalence classes, so that for each $i \in e = \{0, \dots, e-1\}$, we denote by R_i the i -th equivalence class of R . Then, define $f: e \rightarrow n$ as the equivalence on $e+n$

$$R \cup \{(i, j) \mid i \in e \text{ and } j \in R_i\} \cup \{(i, j) \mid j \in e \text{ and } i \in R_j\}.$$

It is immediate to see that f satisfies (3) and (4) and that $\boxed{R} = \boxed{f} \bullet$.

► **Remark 17.** A similar property, *functional completeness*, was already considered in [8]. The important difference is that we don't require f to be mono. Ours is a more general notion: every functionally complete cartesian bicategory also has enough maps.

► **Lemma 18.** *If a cartesian bicategory has enough maps, then for every morphism $R: X \rightarrow Y$, there are maps $f: Z \rightarrow X$ and $g: Z \rightarrow Y$ such that*

$$\boxed{R} = \boxed{f} \boxed{g}$$

We call this a comap-map factorisation of R .

Proof. Since there are enough maps, there is a map $h: Z \rightarrow X \otimes Y$ such that

$$\boxed{R} \bullet = \boxed{h}$$

Let $\boxed{f} = \boxed{h}$ and $\boxed{g} = \boxed{h}$, then

$$\boxed{h} = \boxed{h} \bullet \bullet = \bullet \boxed{h} \bullet = \bullet \boxed{f} \bullet \bullet = \bullet \boxed{g} \bullet$$

where the step marked $*$ uses coherence of the comonoid and the fact that h is a map. Therefore we have

$$\boxed{R} = \boxed{R} \bullet \bullet = \bullet \boxed{f} \bullet \bullet = \bullet \boxed{g} \bullet \bullet = \boxed{f} \boxed{g}$$



► **Proposition 19.** *A cartesian bicategory with enough maps satisfies (AC) iff surjective maps split.*

Proof. By Proposition 14, it suffices to prove that (AC) holds if surjective maps split. So let $R: X \rightarrow Y$ be a total relation and take a comap-map factorisation $\boxed{R} = \boxed{f} \boxed{g}$ with maps f, g . Since R is total,

$$\bullet = \boxed{f} \boxed{g} \bullet = \boxed{f} \bullet$$

so f is surjective. Since surjective maps split, there exists a map h that is a pre-inverse of f , so $h; f = \text{id}$. Then

$$\boxed{h} \leq \boxed{h} \boxed{f} \boxed{f} = \boxed{f}$$

and therefore $\boxed{h} \boxed{g} \leq \boxed{f} \boxed{g} = \boxed{R}$, so R contains a map.

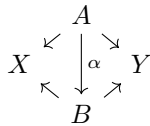


3 The Span[~] construction

In [5], we used a construction that, for any category with finite limits, gives rise to a cartesian bicategory. Since it plays a key role in our main result (Theorem 30), we recall the construction and extend it to arbitrary categories with finite products and *weak* pullbacks.

We start by recalling the standard notion of bicategory of spans.

► **Definition 20** (Span). Let \mathcal{C} be a finitely complete category. A *span* from X to Y is a pair of arrows $X \leftarrow A \rightarrow Y$ in \mathcal{C} . A morphism $\alpha: (X \leftarrow A \rightarrow Y) \Rightarrow (X \leftarrow B \rightarrow Y)$ is an arrow $\alpha: A \rightarrow B$ in \mathcal{C} s.t. the diagram below commutes. Spans $X \leftarrow A \rightarrow Y$ and $X \leftarrow B \rightarrow Y$ are *isomorphic* if α is an isomorphism. For $X \in \mathcal{C}$, the *identity span* is $X \xleftarrow{\text{id}_X} X \xrightarrow{\text{id}_X} X$. The composition of $X \leftarrow A \xrightarrow{f} Y$ and $Y \xleftarrow{g} B \rightarrow Z$ is $X \leftarrow A \times_{f,g} B \rightarrow Z$, obtained by taking the pullback of f and g . This data defines the bicategory [1] $\text{Span}(\mathcal{C})$: the objects are those of \mathcal{C} , the arrows are spans and 2-cells are homomorphisms. Finally, $\text{Span}(\mathcal{C})$ has monoidal product given by the product in \mathcal{C} , with unit the final object $1 \in \mathcal{C}$.



To avoid the complications of non-associative composition, it is common to consider a *category* of spans, where isomorphic spans are equated: let $\text{Span}^{\leq} \mathcal{C}$ be the monoidal category that has isomorphism classes of cospans as arrows. Note that, when going from bicategory to category, after identifying isomorphic arrows it is usual to simply discard the 2-cells. Differently, we consider $\text{Span}^{\leq} \mathcal{C}$ to be locally preordered with $(X \leftarrow A \rightarrow Y) \leq (X \leftarrow B \rightarrow Y)$ if there exists a morphism $\alpha: (X \leftarrow A \rightarrow Y) \Rightarrow (X \leftarrow B \rightarrow Y)$. It is an easy exercise to verify that this (pre)ordering is well-defined and compatible with composition and monoidal product. Note that, in general, \leq is a genuine preorder: i.e. it is possible that $(X \rightarrow A \leftarrow Y) \leq (X \rightarrow B \leftarrow Y) \leq (X \rightarrow A \leftarrow Y)$ without the cospans being isomorphic.

Since $\text{Span}^{\leq} \mathcal{C}$ is preorder enriched, rather than poset enriched, it is *not* a cartesian bicategory. However, one can transform a preorder enriched category into a poset enriched one with a simple construction: for $\text{Span}^{\leq} \mathcal{C}$, one first defines $\sim = \leq \cap \geq$, namely $(X \leftarrow A \rightarrow Y) \sim (X \leftarrow B \rightarrow Y)$ iff there exists $\alpha: (X \leftarrow A \rightarrow Y) \Rightarrow (X \leftarrow B \rightarrow Y)$ and $\beta: (X \leftarrow B \rightarrow Y) \Rightarrow (X \leftarrow A \rightarrow Y)$, and then one takes equivalence classes of morphisms of $\text{Span}^{\leq} \mathcal{C}$ modulo \sim . It is worth observing that pullbacks are no longer necessary to compose \sim -equivalence classes of spans: weak pullbacks are sufficient, since non-isomorphic weak pullbacks of the same cospan all belong to the same \sim -equivalence class.

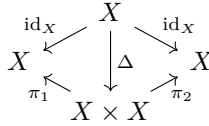
► **Definition 21** (Span[~]). Let \mathcal{C} be a category with finite products and weak pullbacks. The posetal category $\text{Span}^{\sim} \mathcal{C}$ has the same objects as \mathcal{C} and as morphisms \sim -equivalence classes of spans. The order is defined as in $\text{Span}^{\leq} \mathcal{C}$. Composition is given by weak pullbacks in \mathcal{C} . Identities, monoidal product and unit are as in $\text{Span}(\mathcal{C})$.

Like in **Rel**, the comonoid structure is given for any object X by the diagonal and final morphism in \mathcal{C} : $\overset{X}{\curvearrowright} \bullet$ is the span $X \leftarrow X \rightarrow X \times X$ and $\overset{X}{\bullet}$ is $X \leftarrow X \rightarrow 1$.

► **Proposition 22.** *Let \mathcal{C} be a category with finite products and weak pullbacks. Then $\text{Span}^{\sim} \mathcal{C}$ is a cartesian bicategory.*

Proof. For $\overset{X}{\curvearrowright} \bullet$ we take the span $X \times X \leftarrow X \rightarrow X$ and for $\bullet \overset{X}{\curvearrowright}$ we take $1 \leftarrow X \rightarrow X$.

With this information, one has only to check that the inequalities in Definition 1 hold: each of them is witnessed by a commutative diagrams in \mathcal{C} . As an example, we illustrate $\text{---}_X \leq \text{---}_X \bullet \bullet_X$. The left hand side is the span $X \xleftarrow{\text{id}_X} X \xrightarrow{\text{id}_X} X$. The right hand side is the composition of $X \xleftarrow{\text{id}_X} X \xrightarrow{!} 1$ and $1 \xleftarrow{!} X \xrightarrow{\text{id}_X} X$. Since the product $X \xleftarrow{\pi_1} X \times X \xrightarrow{\pi_2} X$ is a pullback of $X \xrightarrow{!} 1 \xleftarrow{!} X$, the composition turns out to be exactly the span $X \xleftarrow{\pi_1} X \times X \xrightarrow{\pi_2} X$. Now the diagonal $\Delta: X \rightarrow X \times X$ makes the following diagram in \mathcal{C} commute. Therefore Δ witnesses the inequality $\text{---}_X \leq \text{---}_X \bullet \bullet_X$.



The following is a characterisation of $\text{Span}^{\sim} \mathcal{C}$ maps: a span $X \leftarrow A \rightarrow Y$ is a map iff it is \sim -equivalent to $X \xleftarrow{\text{id}_X} X \xrightarrow{f} Y$ for some f in \mathcal{C} . Moreover it is surjective iff f is a split epi.

► **Proposition 23.** *Let \mathcal{C} be a category with finite products and weak pullbacks. Then $\text{Map}(\text{Span}^{\sim} \mathcal{C}) \cong \mathcal{C}$ and surjective maps in $\text{Span}^{\sim} \mathcal{C}$ are exactly split epis in \mathcal{C} .*

Proof. Since \mathcal{C} has finite products, it is endowed with a cartesian monoidal structure. This means in particular that $\text{---} \bullet = \text{---} \boxed{g} \bullet$ for all g in \mathcal{C} .

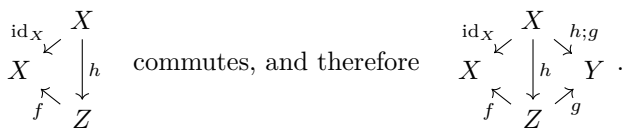
Let $F: \mathcal{C} \rightarrow \text{Span}^{\sim} \mathcal{C}$ be the identity on objects and mapping a morphism $f: X \rightarrow Y$ to the span $X \xleftarrow{\text{id}_X} X \xrightarrow{f} Y$. It is easy to check that F is a monoidal functor.

Since every morphism in \mathcal{C} is a comonoid homomorphism, F factors as $\mathcal{C} \xrightarrow{F'} \text{Map}(\text{Span}^{\sim} \mathcal{C}) \rightarrow \text{Span}^{\sim} \mathcal{C}$. To conclude that F' is an isomorphism, it is enough to show that every $\text{Span}^{\sim} \mathcal{C}$ map is the \sim -equivalence class of some span $X \xleftarrow{\text{id}_X} X \xrightarrow{f} Y$.

Now, if $X \xleftarrow{f} Z \xrightarrow{g} Y$ is a map in $\text{Span}^{\sim} \mathcal{C}$, in particular

$$\text{---} \bullet \leq \text{---} \boxed{f} \boxed{g} \bullet = \text{---} \boxed{f} \bullet$$

Therefore, by Definition of the ordering in $\text{Span}^{\sim} \mathcal{C}$, there is a morphism $h: X \rightarrow Z$ such that



The two spans are thus equal in $\text{Span}^{\sim} \mathcal{C}$, since they are both maps.

We can now prove the second part of the proposition. If $\pi: X \rightarrow Y$ is a map in \mathcal{C} such that $F(\pi)$ is surjective in $\text{Span}^{\sim} \mathcal{C}$, then we have

$$\bullet \text{---} \leq \bullet \text{---} \boxed{\pi} \text{---} \quad \text{and therefore there is } \iota: Y \rightarrow X \text{ such that}$$

$$\begin{array}{ccc}
 Y & & \\
 \downarrow \iota & \searrow \text{id}_Y & \\
 & Y & \\
 \downarrow & \nearrow \pi & \\
 X & &
 \end{array}$$

so π is a split epi. The converse direction is obvious.

► **Proposition 24.** *$\text{Span}^{\sim} \mathcal{C}$ has enough maps.*

15:12 The Axiom of Choice in Cartesian Bicategories

Proof. In a cartesian bicategory, for all $R: X \rightarrow I$ we have $R \leq \dashv\!\!-\!x \bullet$. In the special case when R is a map $g: X \rightarrow I$, by Proposition 8, it holds that $g = \dashv\!\!-\!x \bullet$. Now take a morphism $R: X \rightarrow I$ in $\text{Span}^{\sim} \mathcal{C}$. By definition, R is a span $X \xleftarrow{f} A \xrightarrow{g} I$. Observe that by Proposition 23, both f and g are maps in $\text{Span}^{\sim} \mathcal{C}$. Therefore $g = \dashv\!\!-\!x \bullet$ and $\text{Span}^{\sim} \mathcal{C}$ has enough maps. \blacktriangleleft

By Proposition 19, the two propositions above entail the following.

► **Corollary 25.** $\text{Span}^{\sim} \mathcal{C}$ satisfies (AC).

► **Example 26.** Let \mathbf{FinSet} be the category with natural numbers as objects and as morphisms functions (as in Example 9, natural numbers are regarded as finite sets). The category $\text{Span}^{\sim} \mathbf{FinSet}^{\text{op}} = \text{Cospan}^{\sim} \mathbf{FinSet}$ satisfies (AC) by Corollary 25. This category is particularly relevant for different reasons. First, it is the cartesian bicategory on one object (see [5, Theorem 31]) or, using the terminology in [4], it is the Carboni-Walters category freely generated by the empty Frobenius theory. Moreover, after forgetting its posetal enrichment, it is the PROP \mathbf{Frob} of special Frobenius bimonoids which appears to be of fundamental importance in several works (e.g. in [20, 2]). Finally, the cartesian bicategory of equivalence relations, \mathbf{ERel} from Example 9, can be obtained as quotient of $\text{Cospan}^{\sim} \mathbf{FinSet}$: to pass from cospans to equivalence relations, it suffices to equate

$$\bullet \longrightarrow \bullet = \boxed{}.$$

Since \mathbf{ERel} does not satisfy (AC), by Corollary 25, there is no category \mathcal{C} , such that \mathbf{ERel} is $\text{Span}^{\sim} \mathcal{C}$. Instead, \mathbf{PERel} can be put in Span^{\sim} form: it is $\text{Span}^{\sim} \mathbf{FinSet}_p^{\text{op}} = \text{Cospan}^{\sim} \mathbf{FinSet}_p$ for \mathbf{FinSet}_p being defined as \mathbf{FinSet} but with partial functions as morphisms. Indeed, as we will see in the next section, any cartesian bicategory with enough maps that satisfies (AC) arises from the Span^{\sim} construction.

4 Characterising cartesian bicategories with choice

In this section, we prove our characterisation result (Theorem 30). First, we observe that (AC) allows us to construct maps witnessing certain inequalities.

► **Lemma 27.** Let \mathcal{B} be a cartesian bicategory with choice and

$$\begin{array}{ccc} & A & \\ f \swarrow & & \searrow g \\ B & & C \\ h \swarrow & D & \nearrow k \end{array}$$

a diagram of maps such that $\boxed{f} \boxed{g} \leq \boxed{h} \boxed{k}$. Then there is a map $\omega: A \rightarrow D$ (called witness) such that the following diagram commutes.

$$\begin{array}{ccc} & A & \\ f \swarrow & \downarrow \omega & \searrow g \\ B & & C \\ h \swarrow & D & \nearrow k \end{array}$$

Proof. Consider $R: A \rightarrow D$ given by

$$\boxed{R} = \begin{array}{c} \bullet \\ \swarrow \quad \searrow \\ \boxed{f} \quad \boxed{h} \\ \downarrow \quad \downarrow \\ \boxed{g} \quad \boxed{k} \\ \swarrow \quad \searrow \\ \bullet \end{array}$$

One readily checks that $\boxed{R} \leq \boxed{f} \boxed{h}$ and $\boxed{R} \leq \boxed{g} \boxed{k}$.
 R is total, since

$$\begin{array}{c} \bullet \\ \swarrow \quad \searrow \\ \boxed{f} \quad \boxed{h} \\ \downarrow \quad \downarrow \\ \boxed{g} \quad \boxed{k} \\ \swarrow \quad \searrow \\ \bullet \end{array} = \begin{array}{c} \bullet \\ \swarrow \quad \searrow \\ \boxed{f} \quad \boxed{h} \quad \boxed{k} \\ \downarrow \quad \downarrow \\ \boxed{g} \\ \swarrow \quad \searrow \\ \bullet \end{array} \geq \begin{array}{c} \bullet \\ \swarrow \quad \searrow \\ \boxed{f} \quad \boxed{f} \quad \boxed{g} \\ \downarrow \quad \downarrow \\ \boxed{g} \\ \swarrow \quad \searrow \\ \bullet \end{array} \geq \begin{array}{c} \bullet \\ \swarrow \quad \searrow \\ \boxed{g} \\ \downarrow \\ \bullet \end{array} \geq \bullet$$

so by the axiom of choice, there is a map $\omega \leq R$. This satisfies

$$\boxed{\omega} \boxed{h} \leq \boxed{R} \boxed{h} \leq \boxed{f} \boxed{h} \boxed{h} \leq \boxed{f}$$

and

$$\boxed{\omega} \boxed{k} \leq \boxed{R} \boxed{k} \leq \boxed{g} \boxed{k} \boxed{k} \leq \boxed{g}$$

and since both side are maps we have equality by Proposition 8. \blacktriangleleft

► **Lemma 28.** Let \mathcal{B} be a cartesian bicategory and consider the following diagram in $\text{Map } \mathcal{B}$.

$$\begin{array}{ccc} & A & \\ f \swarrow & & \searrow g \\ B & & C \\ h \searrow & & \swarrow k \\ & D & \end{array} \tag{5}$$

1. $\boxed{f} \boxed{g} \leq \boxed{h} \boxed{k}$ if and only if (5) commutes.
2. If \mathcal{B} has choice and $\boxed{f} \boxed{g} = \boxed{h} \boxed{k}$, then (5) is a weak pullback.
3. If \mathcal{B} has choice and enough maps then $\boxed{f} \boxed{g} = \boxed{h} \boxed{k}$ iff (5) is a weak pullback.

Proof. 1. If the diagram commutes, then

$$\boxed{f} \boxed{g} \leq \boxed{f} \boxed{g} \boxed{k} \boxed{k} = \boxed{f} \boxed{f} \boxed{h} \boxed{k} \leq \boxed{h} \boxed{k}$$

Conversely, if $\boxed{f} \boxed{g} \leq \boxed{h} \boxed{k}$, then

$$\boxed{g} \boxed{k} \leq \boxed{f} \boxed{f} \boxed{g} \boxed{k} \leq \boxed{f} \boxed{h} \boxed{k} \boxed{k} \leq \boxed{f} \boxed{h}$$

and since both sides are maps, they are equal by Proposition 8.

2. Let now \mathcal{B} satisfy (AC) and $\boxed{f} \boxed{g} = \boxed{h} \boxed{k}$. We want to show that (5) is a weak pullback. Given a commutative diagram of solid arrows below,

$$\begin{array}{ccc} & T & \\ b \swarrow & \downarrow \omega & \searrow c \\ B & A & C \\ f \swarrow & & \searrow g \\ & D & \end{array}$$

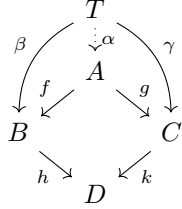
we need to construct the dotted arrow. By Lemma 28.1, we get

$$\boxed{b} \boxed{c} \leq \boxed{h} \boxed{k} = \boxed{f} \boxed{g}$$

and therefore by Lemma 27 we get $\omega: T \rightarrow A$ as desired.

15:14 The Axiom of Choice in Cartesian Bicategories

3. Let now \mathcal{B} have also enough maps and let (5) be a weak pullback. By Lemma 28.1, it suffices to prove that $-(h) \circ (k) \leq -(f) \circ (g)$. By Lemma 18, take $-(h) \circ (k) = -(\beta) \circ (\gamma)$ to be a factorisation with $\beta: T \rightarrow B$ and $\gamma: T \rightarrow C$. By Lemma 28.1, the external square of the following diagram commutes.



Since (5) is a weak pullback, there is $\alpha: T \rightarrow A$ making the above commute. With this we get

$$-(h) \circ (k) = -(\beta) \circ (\gamma) = -(f) \circ (\alpha) \circ (\alpha) \circ (g) \leq -(f) \circ (g) \quad \blacktriangleleft$$

By the third point of the above lemma and Lemma 18 we immediately get the following.

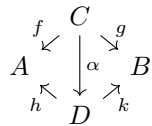
► **Corollary 29.** *Let \mathcal{B} be a cartesian bicategory with enough maps and choice. Then $\text{Map}(\mathcal{B})$ has weak pullbacks given by the comap-map factorisation.*

We can now state our main result.

► **Theorem 30.** *Let \mathcal{B} be a cartesian bicategory with enough maps. \mathcal{B} satisfies (AC) if and only if there is a category \mathcal{C} with products and weak pullbacks such that $\mathcal{B} \cong \text{Span}^{\sim} \mathcal{C}$. More precisely, (AC) holds if and only if there exists a functor $F: \text{Span}^{\sim} \text{Map}(\mathcal{B}) \rightarrow \mathcal{B}$ that is an isomorphism.*

Proof. By Corollary 25, $\text{Span}^{\sim} \text{Map}(\mathcal{B})$ satisfies (AC), so if there is an isomorphism F also \mathcal{B} satisfies (AC).

Now assume that \mathcal{B} has enough maps and satisfies (AC). Since every morphism in $\text{Map}(\mathcal{B})$ is a comonoid homomorphism, $\text{Map}(\mathcal{B})$ is a cartesian monoidal category and hence has finite products, see [8, Theorem 1.6]. It furthermore has weak pullbacks by Corollary 29. We define $F: \text{Span}^{\sim} \text{Map}(\mathcal{B}) \rightarrow \mathcal{B}$ to be the identity on objects and mapping a span $X \xleftarrow{f} Z \xrightarrow{g} Y$ into the composite $f^{\text{op}} ; g$ in \mathcal{B} . To prove that F preserves the ordering, observe that if



is a commutative diagram in $\text{Map}(\mathcal{B})$, then

$$-(f) \circ (g) = -(h) \circ (\alpha) \circ (\alpha) \circ (k) \leq -(h) \circ (k)$$

That F indeed preserves composition follows from the weak pullback being given by comap-map factorisation (Corollary 29). The functor F is identity-on-objects and full by Lemma 18. By Lemma 27, the functor reflects the ordering. Therefore it is faithful, hence an equivalence. \blacktriangleleft

5 Related work

Another common example of cartesian bicategories, considered in [8], is the category of relations of a regular category. The following definitions can be found in [7].

► **Definition 31.** Let \mathcal{C} be a category. A kernel pair of a morphism $f: X \rightarrow Y$ is a pair of $p_1, p_2: P \rightarrow X$ such that the diagram below is a pullback. An epimorphism is regular if it is the coequaliser of some pair of morphisms. \mathcal{C} is regular if it has finite limits, coequalisers of kernel pairs and regular epis are stable under pullback.

$$\begin{array}{ccc} P & \xrightarrow{p_1} & X \\ p_2 \downarrow & & \downarrow f \\ X & \xrightarrow{f} & Y \end{array}$$

Regular categories admit a well-behaved factorisation system, where every morphism factors as a regular epi followed by a mono. The factorisation is used to define the cartesian bicategory of relations of a regular category.

► **Definition 32.** Given a regular category \mathcal{C} , let $\mathbf{Rel}(\mathcal{C})$ be the category with the same objects as \mathcal{C} and morphisms $X \rightarrow Y$ jointly mono spans, i.e. spans $X \xleftarrow{f} A \xrightarrow{g} Y$ such that the induced map $A \xrightarrow{\langle f, g \rangle} X \times Y$ is mono. For an arbitrary span, $X \xleftarrow{f} A \xrightarrow{g} Y$, its *image* is the jointly mono span given by taking the regular epi-mono factorisation of $A \xrightarrow{\langle f, g \rangle} X \times Y$. The composition of two jointly mono spans is given by first composing them as spans via pullback and then taking the image of the resulting span. The identity $X \rightarrow X$ is given by the jointly mono span $X \xleftarrow{\text{id}_X} X \xrightarrow{\text{id}_X} X$. Similar to $\mathbf{Span}^{\sim} \mathcal{C}$, the categorical product of \mathcal{C} induces a monoidal product on $\mathbf{Rel}(\mathcal{C})$. Furthermore, the ordering is defined as for $\mathbf{Span}^{\sim} \mathcal{C}$: $(X \leftarrow A \rightarrow Y) \leq (X \leftarrow B \rightarrow Y)$ if there exists a morphism of spans $\alpha: (X \leftarrow A \rightarrow Y) \Rightarrow (X \leftarrow B \rightarrow Y)$.

Since $\mathbf{Rel}(\mathcal{C})$ is a cartesian bicategory [8, Example 1.4], it is important to compare the $\mathbf{Rel}(\mathcal{C})$ and $\mathbf{Span}^{\sim} \mathcal{C}$ constructions. In general the two do not coincide. To see this, it is enough to take $\mathbf{FinSet}^{\text{op}}$: $\mathbf{Rel}(\mathbf{FinSet}^{\text{op}})$ is \mathbf{ERel} which, as discussed in Example 26, is a proper quotient of $\mathbf{Span}^{\sim} \mathbf{FinSet}^{\text{op}}$. This is an instance of a more general fact:

► **Proposition 33.** *There is a full monoidal functor $F: \mathbf{Span}^{\sim} \mathcal{C} \rightarrow \mathbf{Rel}(\mathcal{C})$ given by mapping a span to its image.*

The proof of the above and the following statements can be found in the Appendix.

► **Lemma 34.** $\mathbf{Rel}(\mathcal{C})$ has enough maps.

► **Proposition 35.** $\mathbf{Span}^{\sim} \mathcal{C} \cong \mathbf{Rel}(\mathcal{C})$ if and only if (AC) holds in $\mathbf{Rel}(\mathcal{C})$.

It is known that surjective maps in $\mathbf{Rel}(\mathcal{C})$ are precisely regular epis in \mathcal{C} , see [8, Theorem 3.5]. Using Proposition 19, we have the following.

► **Corollary 36.** $\mathbf{Span}^{\sim} \mathcal{C} \cong \mathbf{Rel}(\mathcal{C})$ iff regular epis split in \mathcal{C} .

References

- 1 Jean Bénabou. Introduction to bicategories. In *Reports of the Midwest Category Seminar*, pages 1–77. Springer, 1967.
- 2 Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Paweł Sobociński, and Fabio Zanasi. Rewriting modulo symmetric monoidal structure. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 710–719. ACM, 2016.

15:16 The Axiom of Choice in Cartesian Bicategories

- 3 Filippo Bonchi, Joshua Holland, Dusko Pavlovic, and Pawel Sobocinski. Refinement for signal flow graphs. In *28th International Conference on Concurrency Theory (CONCUR 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.
- 4 Filippo Bonchi, Dusko Pavlovic, and Pawel Sobocinski. Functorial Semantics for Relational Theories. *CoRR*, abs/1711.08699, 2017. [arXiv:1711.08699](#).
- 5 Filippo Bonchi, Jens Seeber, and Pawel Sobocinski. Graphical Conjunctive Queries. In Dan Ghica and Achim Jung, editors, *27th EACSL Annual Conference on Computer Science Logic (CSL 2018)*, volume 119 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:23, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CSL.2018.13.
- 6 Roberto Bruni and Fabio Gadducci. Some algebraic laws for spans (and their connections with multirelations). In *Proc. RelMiS 2001, ENTCS*, volume 44, 2001.
- 7 Carsten Butz. Regular categories and regular logic. *BRICS Lecture Series LS-98-2*, 1998.
- 8 Aurelio Carboni and Robert FC Walters. Cartesian bicategories I. *Journal of pure and applied algebra*, 49(1-2):11–32, 1987.
- 9 Brandon Coya and Brendan Fong. Corelations are the prop for extraspecial commutative Frobenius monoids. *Theory and Applications of Categories*, 32(11):380–395, 2017.
- 10 Brendan Fong. *The Algebra of Open and Interconnected Systems*. PhD thesis, University of Oxford, 2016.
- 11 Brendan Fong and David I Spivak. Graphical Regular Logic. *arXiv preprint arXiv:1812.05765*, 2018.
- 12 Brendan Fong and Fabio Zanasi. Universal Constructions for (Co)Relations: categories, monoidal categories, and props. *Logical Methods in Computer Science*, Volume 14, Issue 3, September 2018. doi:10.23638/LMCS-14(3:14)2018.
- 13 Thomas Fox. Coalgebras and cartesian categories. *Communications in Algebra*, 4(7):665–667, 1976.
- 14 Peter J Freyd and Andre Scedrov. *Categories, allegories*, volume 39. Elsevier, 1990.
- 15 Ichiro Hasuo and Naohiko Hoshino. Semantics of higher-order quantum computation via geometry of interaction. *Annals of Pure and Applied Logic*, 168(2):404–469, 2017.
- 16 Horst Herrlich. *Axiom of choice*. Springer, 2006.
- 17 Bart Jacobs. *Categorical logic and type theory*, volume 141. Elsevier, 1999.
- 18 Bart Jacobs and Jorik Mandemaker. Coreflections in algebraic quantum logic. *Foundations of physics*, 42(7):932–958, 2012.
- 19 Peter T Johnstone. *Sketches of an elephant: A topos theory compendium*, volume 2. Oxford University Press, 2002.
- 20 Stephen Lack. Composing props. *Theory and Applications of Categories*, 13(9):147–163, 2004.
- 21 Samuel J Mason. Feedback theory-some properties of signal flow graphs. *Proceedings of the IRE*, 41(9):1144–1156, 1953.
- 22 E. Patterson. Knowledge Representation in Bicategories of Relations. *ArXiv e-prints*, June 2017. [arXiv:1706.00526](#).
- 23 Peter Selinger. A survey of graphical languages for monoidal categories. In *New structures for physics*, pages 289–355. Springer, 2010.
- 24 Thomas Streicher. *Semantics of type theory: correctness, completeness and independence results*. Springer Science & Business Media, 2012.
- 25 Fabio Zanasi. *Interacting Hopf Algebras: the theory of linear systems*. PhD thesis, Ecole Normale Supérieure de Lyon, 2015.

A

 Proof of Section 5

Proof of Proposition 33. F preserves the ordering, because of the universal property of the image. Therefore F is well-defined, since equivalent spans in $\mathbf{Span}^{\sim}\mathcal{C}$ are mapped to isomorphic spans in $\mathbf{Rel}(\mathcal{C})$. It is easily verified that F preserves identities and composition. Since monos and regular epis are closed under product, F preserves the monoidal structure. Finally, F is full because any jointly monic span is its own image. ◀

Proof of Lemma 34. By [8, Theorem 3.5], $\mathbf{Rel}(\mathcal{C})$ is functionally complete (see Remark 17). ◀

Proof of Proposition 35. If $\mathbf{Span}^{\sim}\mathcal{C} \cong \mathbf{Rel}(\mathcal{C})$, then $\mathbf{Rel}(\mathcal{C})$ satisfies (AC) by Corollary 25. If on the other hand, $\mathbf{Rel}(\mathcal{C})$ satisfies (AC), then, since it has enough maps, Proposition 19 guarantees that surjective maps split. Now surjective maps in $\mathbf{Rel}(\mathcal{C})$ are regular epis in \mathcal{C} ([8, Theorem 3.5]), hence the latter split in \mathcal{C} . We prove that in that case F – defined and shown to be full in Proposition 33 – is furthermore faithful, for which it suffices to show that it reflects the ordering. So let $A \xleftarrow{f} B \xrightarrow{g} C$ and $A \xleftarrow{f'} B' \xrightarrow{g'} C$ be spans such that the image of the first is included in the image of the second under F . Assume without loss of generality that A is terminal, which we can achieve by bending the input around to the right. Let $B \xrightarrow{\pi} J \xrightarrow{\iota} C$ be a regular epi-mono factorisation of g and likewise for g' . Then there is a morphism $\alpha: J \rightarrow J'$

$$\begin{array}{ccc} B & \xrightarrow{\pi} & J & \xrightarrow{\iota} & C \\ & & \downarrow \alpha & \nearrow \iota' & \\ B' & \xrightarrow{\pi'} & J' & & \end{array}$$

Since π' is regular epi, it splits by the preceding observation, and thus there is $\beta: B \rightarrow B'$ such that

$$\begin{array}{ccc} B & \xrightarrow{g} & C \\ \downarrow \beta & \nearrow g' & \\ B' & & \end{array} .$$

It follows that the inclusion between the spans holds in $\mathbf{Span}^{\sim}\mathcal{C}$. ◀

Linear-Time Graph Algorithms in GP 2

Graham Campbell 

Department of Computer Science, University of York, United Kingdom

<https://gjcampbell.co.uk/>

gjc510@york.ac.uk

Brian Courtehoue 

Department of Computer Science, University of York, United Kingdom

<https://www.cs.york.ac.uk/people/brianc>

bc956@york.ac.uk

Detlef Plump 

Department of Computer Science, University of York, United Kingdom

<https://www-users.cs.york.ac.uk/det/>

detlef.plump@york.ac.uk

Abstract

GP 2 is an experimental programming language based on graph transformation rules which aims to facilitate program analysis and verification. However, implementing graph algorithms efficiently in a rule-based language is challenging because graph pattern matching is expensive. GP 2 mitigates this problem by providing *rooted* rules which, under mild conditions, can be matched in constant time. In this paper, we present linear-time GP 2 programs for three problems: tree recognition, binary directed acyclic graph (DAG) recognition, and topological sorting. In each case, we show the correctness of the program, prove its linear time complexity, and also give empirical evidence for the linear run time. For DAG recognition and topological sorting, the linear behaviour is achieved by implementing depth-first search strategies based on an encoding of stacks in graphs.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Graph transformation, rooted graph programs, GP 2, linear-time algorithms, depth-first search, topological sorting

Digital Object Identifier 10.4230/LIPIcs.CALCO.2019.16

1 Introduction

Rule-based graph transformation was established as a research field in the 1970s and has since then been the subject of countless articles. While many of these contributions have a theoretical nature (see the monograph [8] for a recent overview), there has also been work on languages and tools for executing and analysing graph transformation systems.

Languages based on graph transformation rules include AGG [18], GReAT [1], GROOVE [10], GrGen.Net [13], Henshin [3] and PORGY [9]. This paper focuses on GP 2 [14], an experimental graph programming language which aims to facilitate formal reasoning on programs. The language has a simple formal semantics and is computationally complete in that every computable function on graphs can be programmed [15]. Research on graph programs has provided, for example, a Hoare-calculus for program verification [16, 17] and a static analysis for confluence checking [12].

A challenge for the design and implementation of graph transformation languages is to narrow the performance gap between imperative and rule-based graph programming. The bottleneck for achieving fast graph transformation is the cost of graph matching. In general, matching the left-hand graph L of a rule within a host graph G requires time $\text{size}(G)^{\text{size}(L)}$ (which is polynomial since L is fixed). As a consequence, linear-time imperative graph algorithms may be slowed down to polynomial time when they are recast as rule-based graph programs.



© Graham Campbell, Brian Courtehoue, and Detlef Plump;

licensed under Creative Commons License CC-BY

8th Conference on Algebra and Coalgebra in Computer Science (CALCO 2019).

Editors: Markus Roggenbach and Ana Sokolova; Article No. 16; pp. 16:1–16:23

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

To mitigate this problem, GP 2 supports *rooted* graph transformation which was first proposed by Dörr [7]. The idea is to distinguish certain nodes as *roots* and to match roots in rules with roots in host graphs. Then only the neighbourhood of host graph roots needs to be searched for matches, allowing, under mild conditions, to match rules in constant time. In [5], *fast* rules were identified as a class of rooted rules that can be applied in constant time if host graphs have a bounded node degree and contain a bounded number of roots.

The condition of a bounded number of host graph roots can be satisfied by requiring unrooted input graphs and using in loops only rules that do not increase the number of roots. This simply means that no such rule must have more roots in its right-hand side than in its left-hand side. (A refined condition considers the “root balance” of all rules in a loop body simultaneously.) The condition that host graphs must have a bounded node degree depends on the application domain of a program. For example, traffic networks or digital circuits can be considered as graphs of bounded degree.

The first linear-time graph problem implemented by a GP 2 program with fast rules was 2-colouring. In [6] it is shown that this program colours connected graphs of bounded degree in linear time. The compiled program even matches the speed of Sedgewick’s textbook C program [19] on grid graphs of up to 100,000 nodes.

In this paper, we continue to provide evidence that rooted graph programs can rival the time complexity of graph algorithms (on bounded-degree graphs) in conventional programming languages. We present three new case studies: recognition of trees, recognition of binary DAGs, and topological sorting of acyclic graphs. Each of these problems is solvable in linear time with algorithms in imperative languages. For each problem, we present a GP 2 program with fast rules, show its correctness, and prove its linear time complexity on graphs of bounded node degree. We also give empirical evidence for the linear run time by presenting benchmark results for graphs of up to 100,000 nodes in various graph classes. For DAG recognition and topological sorting, the linear behaviour is achieved by implementing depth-first search strategies based on an encoding of stacks in host graphs.

It is worth noting that rooted rules per se are not a blueprint for imitating algorithms in imperative languages. This is because GP 2 intentionally does not provide access to the graph data structure of its implementation. As a consequence, for example, currently there seems to be no way of traversing arbitrary disconnected graphs with GP 2 in linear time.

2 The Graph Programming Language GP 2

This section briefly introduces GP 2, a non-deterministic language based on graph-transformation rules, first defined in [14]. Up-to-date versions of the syntax and semantics of GP 2 can be found in [4]. The language is implemented by a compiler generating C code [6].

2.1 Graphs, Rules and Programs

GP 2 programs transform input graphs into output graphs, where graphs are directed and may contain parallel edges and loops. Both nodes and edges are labelled with lists consisting of integers and character strings. This includes the special case of items labelled with the empty list which may be considered as “unlabelled”.

The principal programming construct in GP 2 consist of conditional graph transformation rules labelled with expressions. For example, the rule `i0_push` in Figure 6 has two formal parameters of type `list`, a left-hand graph and a right-hand graph which are specified graphically, and a textual condition starting with the keyword `where`.

The small numbers attached to nodes are identifiers, all other text in the graphs consist of labels. Parameters are typed but in this paper we only need the most general type `list` which represents lists with arbitrary values.

Besides carrying expressions, nodes and edges can be *marked* red, green or blue. In addition, nodes can be marked grey and edges can be dashed. For example, rule `i0_push` in Figure 6 contains red and blue nodes and a blue edge. Marks are convenient, among other things, to record visited items during a graph traversal and to encode auxiliary structures in graphs. The programs in the following sections use marks extensively.

Rules operate on *host graphs* which are labelled with constant values (lists containing integers and character strings). Formally, the application of a rule to a host graph is defined as a two-stage process in which first the rule is instantiated by replacing all variables with values of the same type, and evaluating all expressions. This yields a standard rule (without expressions) in the so-called double-pushout approach with relabelling [11]. In the second stage, the instantiated rule is applied to the host graph by constructing two suitable pushouts. We refer to [4] for details and only give an equivalent operational description of rule application.

Applying a rule $L \Rightarrow R$ to a host graph G works roughly as follows: (1) Replace the variables in L and R with constant values and evaluate the expressions in L and R , to obtain an instantiated rule $\hat{L} \Rightarrow \hat{R}$. (2) Choose a subgraph S of G isomorphic to \hat{L} such that the dangling condition and the rule's application condition are satisfied (see below). (3) Replace S with \hat{R} as follows: numbered nodes stay in place (possibly relabelled), edges and unnumbered nodes of \hat{L} are deleted, and edges and unnumbered nodes of \hat{R} are inserted.

In this construction, the *dangling condition* requires that nodes in S corresponding to unnumbered nodes in \hat{L} (which should be deleted) must not be incident with edges outside S . The rule's application condition is evaluated after variables have been replaced with the corresponding values of \hat{L} , and node identifiers of L with the corresponding identifiers of S . For example, the condition `indeg(1) = 0` of rule `i0_push` in Figure 6 requires that node $g(1)$ has no incoming edges, where $g(1)$ is the node in S corresponding to 1.

A program consists of declarations of conditional rules and procedures, and exactly one declaration of a main command sequence, which is a distinct procedure named `Main`. Procedures must be non-recursive, they can be seen as macros. We describe GP 2's main control constructs.

The call of a rule set $\{r_1, \dots, r_n\}$ non-deterministically applies one of the rules whose left-hand graph matches a subgraph of the host graph such that the dangling condition and the rule's application condition are satisfied. The call *fails* if none of the rules is applicable to the host graph.

The command `if C then P else Q` is executed on a host graph G by first executing C on a copy of G . If this results in a graph, P is executed on the original graph G ; otherwise, if C fails, Q is executed on G . The `try` command has a similar effect, except that P is executed on the result of C 's execution.

The loop command `P!` executes the body P repeatedly until it fails. When this is the case, `P!` terminates with the graph on which the body was entered for the last time. The `break` command inside a loop terminates that loop and transfers control to the command following the loop.

In general, the execution of a program on a host graph may result in different graphs, fail, or diverge. The operational semantics of GP 2 defines a semantic function which maps each host graph to the set of all possible outcomes. See, for example, [15].

2.2 Rooted Programs

The bottleneck for efficiently implementing algorithms in a language based on graph transformation rules is the cost of graph matching. In general, to match the left-hand graph L of a rule within a host graph G requires time polynomial in the size of L [5, 6]. As a consequence, linear-time graph algorithms in imperative languages may be slowed down to polynomial time when they are recast as rule-based programs.

To speed up matching, GP 2 supports *rooted* graph transformation where graphs in rules and host graphs are equipped with so-called root nodes. Roots in rules must match roots in the host graph so that matches are restricted to the neighbourhood of the host graph's roots. We draw root nodes using double circles. For example, in the rule `prune` of Figure 2, the node labelled `y` in the left-hand side and the single node in the right-hand side are roots.

A conditional rule $\langle L \Rightarrow R, c \rangle$ is *fast* if (1) each node in L is undirectedly reachable from some root, (2) neither L nor R contain repeated occurrences of list, string or atom variables, and (3) the condition c contains neither an `edge` predicate nor a test $e_1=e_2$ or $e_1!=e_2$ where both e_1 and e_2 contain a list, string or atom variable.

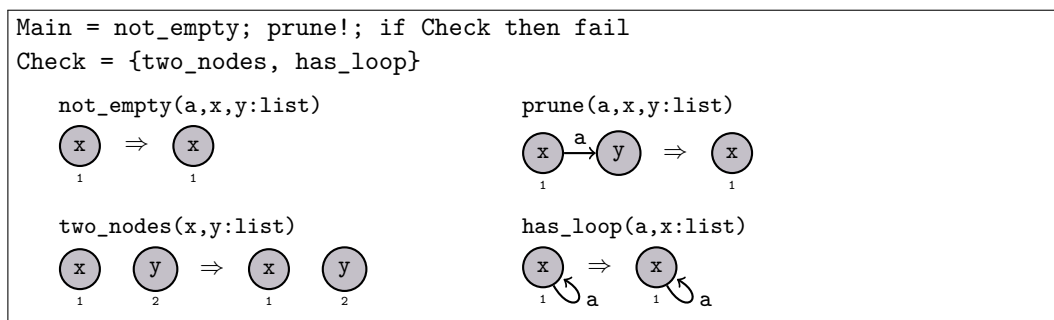
Conditions (2) and (3) will be satisfied by all rules occurring in the following sections; in particular, we neither use the `edge` predicate nor the equality tests. For example, the rules `prune` and `push` in Figure 2 are fast rules.

► **Theorem 1** (Complexity of matching fast rules [5]). *Rooted graph matching can be implemented to run in constant time for fast rules, provided there are upper bounds on the maximal node degree and the number of roots in host graphs.*

When analysing the time complexity of rules and programs, we assume that these are fixed. This is customary in algorithm analysis where programs are fixed and running time is measured in terms of input size [2, 20]. In our setting, the input size is the size of a host graph. The implementation of GP 2 does match fast rooted rules in constant time [6].

3 Recognising Trees

A tree is a graph containing a node from which there is a unique directed path to each node in the graph. It is easy to see that it is possible to generate the collection of all trees by inductively adding new leaf nodes to the discrete graph of size one. Thus, given an input graph, if we prune leaf nodes as long as possible and end up with the discrete graph of size one, then the start graph must have been a tree. Figure 1 is an implementation of this idea in GP 2.



■ **Figure 1** The GP 2 program `is-tree-slow`.

► **Definition 2** (Tree recognition specification). *The tree recognition specification is as follows.*

- Input: An arbitrary labelled graph with every node coloured grey, no root nodes, and no other marks.
- Output: Fail if and only if the input is not a tree.

► **Theorem 3** (Correctness of `is-tree-slow`). *The program `is-tree-slow` fulfills the tree recognition specification.*

Proof. Similar to the proof of Theorem 7. ◀

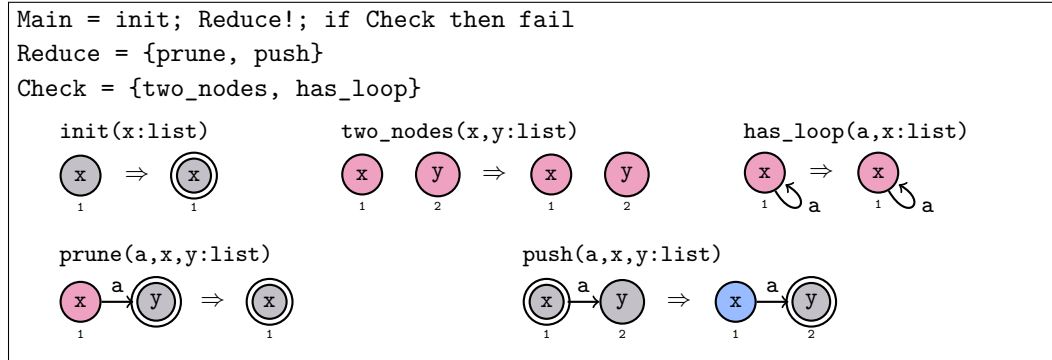
► **Proposition 4** (Termination of `prune!`). *`prune!` terminates after at most $|V_G|$ steps.*

Proof. If $G \Rightarrow H$, then $|V_G| > |V_H|$. Suppose there were an infinite sequence of derivations $G_0 \Rightarrow G_1 \Rightarrow G_2 \Rightarrow \dots$, then there would be an infinite descending chain of natural numbers $|V_{G_0}| > |V_{G_1}| > |V_{G_2}| > \dots$, which contradicts the well-ordering of \mathbb{N} . The last part is immediate since there are only V_G natural numbers less than V_G . ◀

► **Theorem 5** (Complexity of `is-tree-slow`). *Given an input graph of bounded degree, `is-tree-slow` will terminate in quadratic time with respect to the number of nodes in the input graph.*

Proof. Clearly `not_empty` and `Check` run in linear time. Unfortunately `prune` is not a fast rule, and so it takes linear time to find a match. Finding a match for `prune` takes linear time and so by Proposition 4, `prune!` terminates in quadratic time. ◀

Unfortunately, our program does not run in linear time due to our rules not being such that we have constant time matching. We need to modify the program so that we can always perform a match in constant time. Figure 2 is a refined implementation, using root nodes. We will see that this program is not only correct, but always terminates in linear time.



■ **Figure 2** The GP 2 program `is-tree`.

► **Proposition 6** (Correctness of `Reduce!`). *Let G be a rooted input tree and $G \Rightarrow_{Reduce}^* H$. Then, either $|V_H| = 1$ or H is not in normal form.*

Proof. By Lemma 17, $|V_H| \geq 1$. If $|V_G| = 1$, then G is in normal form. Otherwise, either the root node has no children, or it has at least one grey child. In the first case, `prune` must be applicable, and in the second, `push`. Suppose $G \Rightarrow_{Reduce}^* H$. If $|V_H| = 1$, then H is in normal form by the proof to Lemma 17. Otherwise, by Lemma 16 H is a tree and $|V_H| > 1$. Now, the root-node in H (Lemma 17) must have a non-empty neighbourhood. If it has no children, then `prune` must be applicable. Otherwise, `push` must be applicable, since by Corollary 19, there must be a grey node child. So H is not in normal form. ◀

► **Theorem 7** (Correctness of *is-tree*). *The program is-tree fulfills the tree recognition specification.*

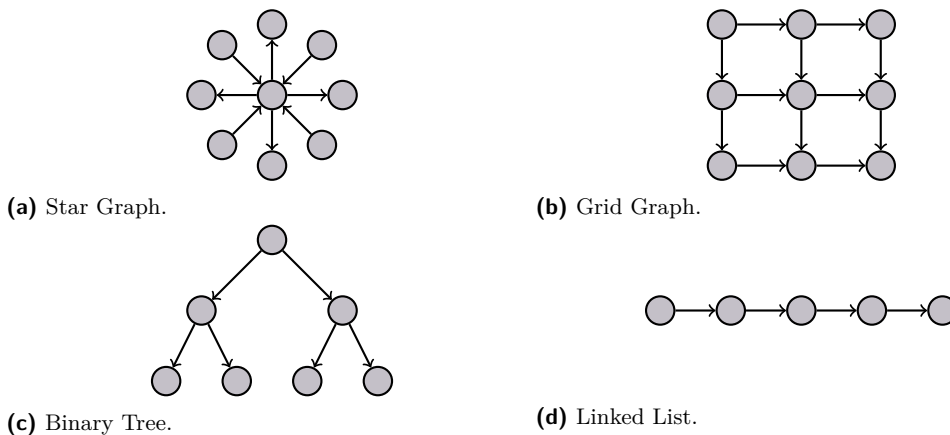
Proof. The *init* rule will fail if the input graph is empty, otherwise, it will make exactly one node rooted. The *Reduce!* step derives the singleton discrete graph if and only if the input was a tree (Proposition 6 and Lemma 16). Finally, by Lemma 17, *Reduce!* cannot derive the empty graph, so it is sufficient for *Check* to test if there is more than one node, or a loop edge. ◀

► **Proposition 8** (Termination of *Reduce!*). *Reduce! terminates after at most $2|V_G|$ steps.*

Proof. Let $\#G$ be the number of nodes, and $\square G$ be the number of grey nodes. If $G \Rightarrow_{prune} H$, then $\#G > \#H$ and $\square G > \square H$. If $G \Rightarrow_{push} H$ then $\#G = \#H$ and $\square G > \square H$. Suppose there were an infinite sequence of derivations $G_0 \Rightarrow G_1 \Rightarrow G_2 \Rightarrow \dots$, then there would be an infinite descending chain of natural numbers $\#G_0 + \square G_0 > \#G_1 + \square G_1 > \#G_2 + \square G_2 > \dots$, which contradicts the well-ordering of \mathbb{N} . To see the last part, notice that $\square G \leq \#G$ for all graphs G , so the result is immediate since there are only $2\#G$ natural numbers less than $2\#G$. ◀

► **Theorem 9** (Complexity of *is-tree*). *Given an input graph of bounded degree, is-tree will terminate in linear time with respect to the number of nodes in the input graph.*

Proof. Clearly *init* and *Check* run in linear time. Since *push* and *prune* are fast rules, they take only constant time (Theorem 1), and then by Proposition 8, *Reduce* can only be applied a linear number of times. Thus, *Reduce!* terminates in linear time too. ◀

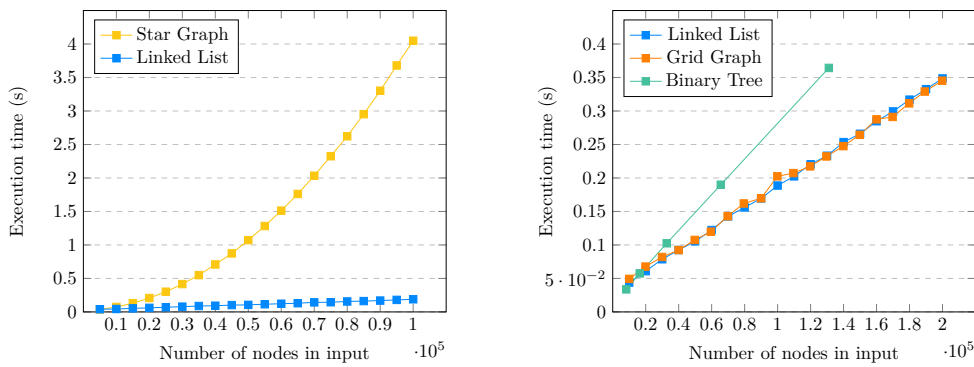


■ **Figure 3** Types of Graph.

We have performed empirical benchmarking to verify the complexity of the program, testing it with Linked Lists, Binary Trees, Grid Graphs, and Star Graphs (Figure 3). Star Graphs are not of bounded degree, so we saw quadratic time complexity as expected. The other graphs are of bounded degree, thus we observed linear time complexity (Figure 4).

4 Recognising Binary DAGs

A *directed acyclic graph* (DAG) is a graph containing no directed cycles. A DAG is *binary* if each of its nodes has an outdegree of at most two.



(a) Star Graphs and Linked Lists.

(b) Bounded Degree Input Graphs.

■ **Figure 4** Measured performance of `is-tree`.

```
Main = try SearchIndeg0Nodes then (if nonempty_stack then skip else fail;
ReduceIndeg0Nodes); if anything then fail
```

| | |
|---|---------------------------------------|
| <p><code>nonempty_stack (x:list)</code></p> | <p><code>anything (x:list)</code></p> |
|---|---------------------------------------|

■ **Figure 5** The GP 2 Program `is-bin-dag`.

`SearchIndeg0Nodes` and `ReduceIndeg0Nodes` are defined in Subsection 4.1. The idea behind recognising connected binary DAGs is as follows. First, using `SearchIndeg0Nodes`, all indegree-0 nodes of the input graph are identified. Then, in `ReduceIndeg0Nodes`, if any indegree-0 nodes have been found, one of them is deleted, and all of its children that become a new indegree-0 node get designated as such. This is repeated until no indegree-0 nodes are left. Every time an indegree-0 node is checked, the number of its children are checked as well. If there are any leftover nodes (i.e. nodes that never had indegree-0 in the execution), then there were no directed cycles, and the input graph is a DAG.

► **Theorem 10** (Correctness of `is-bin-dag`). *The program `is-bin-dag` fulfills the following specification.*

- Input: *A connected graph G with grey unrooted nodes and unmarked edges.*
- Output: *The empty graph if G is a binary DAG, and failure otherwise.*

Proof. If G is the empty graph, a DAG, `SearchIndeg0Nodes` fails by Proposition 11, `anything` does not match, and the output is the empty graph. So assume G is non-empty.

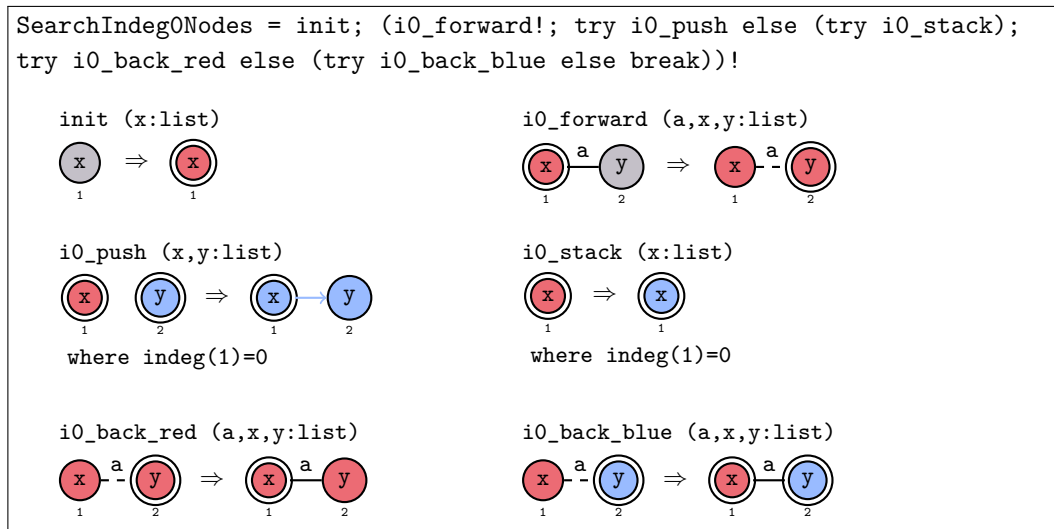
If G has no indegree-0 nodes, `SearchIndeg0Nodes` succeeds by Proposition 11 and does not mark any nodes blue. So `nonempty_stack` will not match, and `fail` will be invoked. So assume G has indegree-0 nodes.

Then Propositions 11 and 12 can be applied to deduce the following. `SearchIndeg0Nodes` succeeds, `nonempty_stack` matches, then `ReduceIndeg0Nodes` gets applied. If G is a binary DAG, the host graph becomes the empty graph, `anything` will not match, and the output is the empty graph. If G is not a binary DAG, there's failure, or a non-empty graph which results in failure since `anything` is matched. ◀

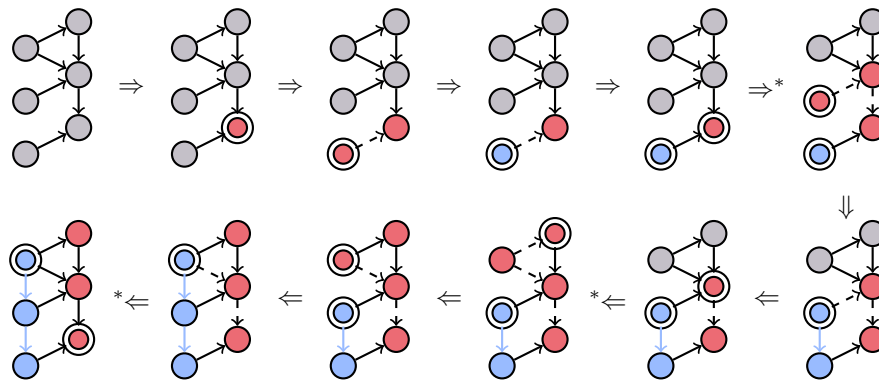
4.1 Correctness of Procedures

The proof of Theorem 10 depends upon the correctness of the procedures `SearchIndeg0Nodes` and `ReduceIndeg0Nodes`. We will now give their definitions and prove their correctness.

`SearchIndeg0Nodes`, as seen in Figure 6, is an undirected modification of depth first search (DFS) as implemented by Bak and Plump [5] [4], with a few key differences. Using DFS ensures that each node is visited. The blue nodes linked with blue edges are a GP 2 implementation of stacks. The top of the stack is the only blue root, making it accessible in constant time. The rules with bidirectional edges (a GP 2 construct) in Figure 6 are semantically equivalent to a non-deterministic rule set call of two distinct variations of that rule with directed edges. The edges in the right and left hand side of these rules have the same orientation.



■ **Figure 6** The `SearchIndeg0Nodes` procedure.



■ **Figure 7** Example execution of `SearchIndeg0Nodes`.

Between the forward and back steps lies the command sequence `try i0_push else (try i0_stack)`. Its purpose is to push the node currently visited by the DFS if it has `indeg=0`. If the stack is nonexistent, there are no blue nodes, and `i0_push` fails. So the program tries to apply `i0_stack`, turning the node into the initial stack element (if its `indeg` is 0). After

the stack has been created, `i0_push` will always be applicable for indegree-0 nodes.

Since the current node may be marked blue by the stack operations after the previous command sequence has been executed, the back step needs to account for that. Hence the program first tries to apply `i0_back_red`, and if that fails, it tries to apply `i0_back_blue`, an alternate version considering a blue current node. In the latter case, the blue node is rooted since we want to keep accessing the top of the stack in constant time.

► **Proposition 11** (Correctness of `SearchIndeg0Nodes`). *The procedure `SearchIndeg0Nodes` fulfills the following specification.*

- Input: A connected graph G with grey unrooted nodes and unmarked edges.
- Output: If G is the empty graph, then failure. Otherwise, G with all non-indegree-0 nodes marked red, at most one of which is a root; indegree-0 nodes marked blue; and the blue nodes connected via newly created blue edges, forming a linked list, of which the head (no incoming blue edges) is a root.

Proof. If G is empty, `init` cannot match, causing failure. Otherwise, the output conditions are satisfied by Lemmata 21 and 22. ◀

The absence of a red root in the output is an edge case caused by `init` being applied to an indegree-0 node. Because then, either `i0_stack` or `i0_push` will be the last rule that is applied, and the red root becomes a blue root.

The procedure `ReduceIndeg0Nodes` starts by trying to apply `unroot` to get rid of any red roots left over by `SearchIndeg0Nodes`. Then it enters the loop `Reduce!`. The blue root in each iteration shall be called the “top root”. First, the program checks whether the top root has more than two children, i.e. whether its outdegree is greater than three, since the blue stack edge needs to be taken into account. If there are too many, the `fail` statement is invoked.

`nontrivial_stack` checks whether the stack has more than one element. If it does not, `add_bottom` artificially adds a node to the bottom of the stack, in order for the following rules to still match.

Next is a non-deterministic choice of rules that cover every case of the number of children the top root has, and how many of those are indegree-0 nodes. In each case, they pop the top root, and push the children that would have indegree 0 after the deletion. `pop!` serves to pop childless indegree-0 nodes for as long as there are any.

► **Proposition 12** (Correctness of `ReduceIndeg0Nodes`). *Let G be a connected graph with red non-indegree-0 nodes containing at most one root, and blue indegree-0 nodes that are connected with blue edges forming a path graph. The blue node with no incoming blue edges is a root. If G minus the blue edges is a binary DAG, `ReduceIndeg0Nodes` yields the empty graph. Otherwise, it yields a non-empty graph.*

Proof sketch. First consider the case of G minus the blue edges being a binary DAG. Assume, for the sake of a contradiction, that the output of `ReduceIndeg0Nodes` contains a node v . By Lemmata 24 and 25, v cannot have been an indegree-0 node when ignoring blue edges at any point during execution. Furthermore, v must have a parent that never was an indegree-0 node ignoring blue edges, because otherwise it would have been marked blue by one of the rule set call rules. The same argument can then be applied to the parent’s parent, and so on indefinitely. Since the input is finite however, two of these ancestors must be equal, meaning that there is a cycle. This contradicts the input minus the blue edges being a DAG.

Next, assume G is not a DAG. Then it has a directed cycle consisting of consecutive nodes v_1, v_2, \dots, v_n . None of these nodes have indegree 0 ignoring blue edges, so they are

16:10 Linear-Time Graph Algorithms in GP 2

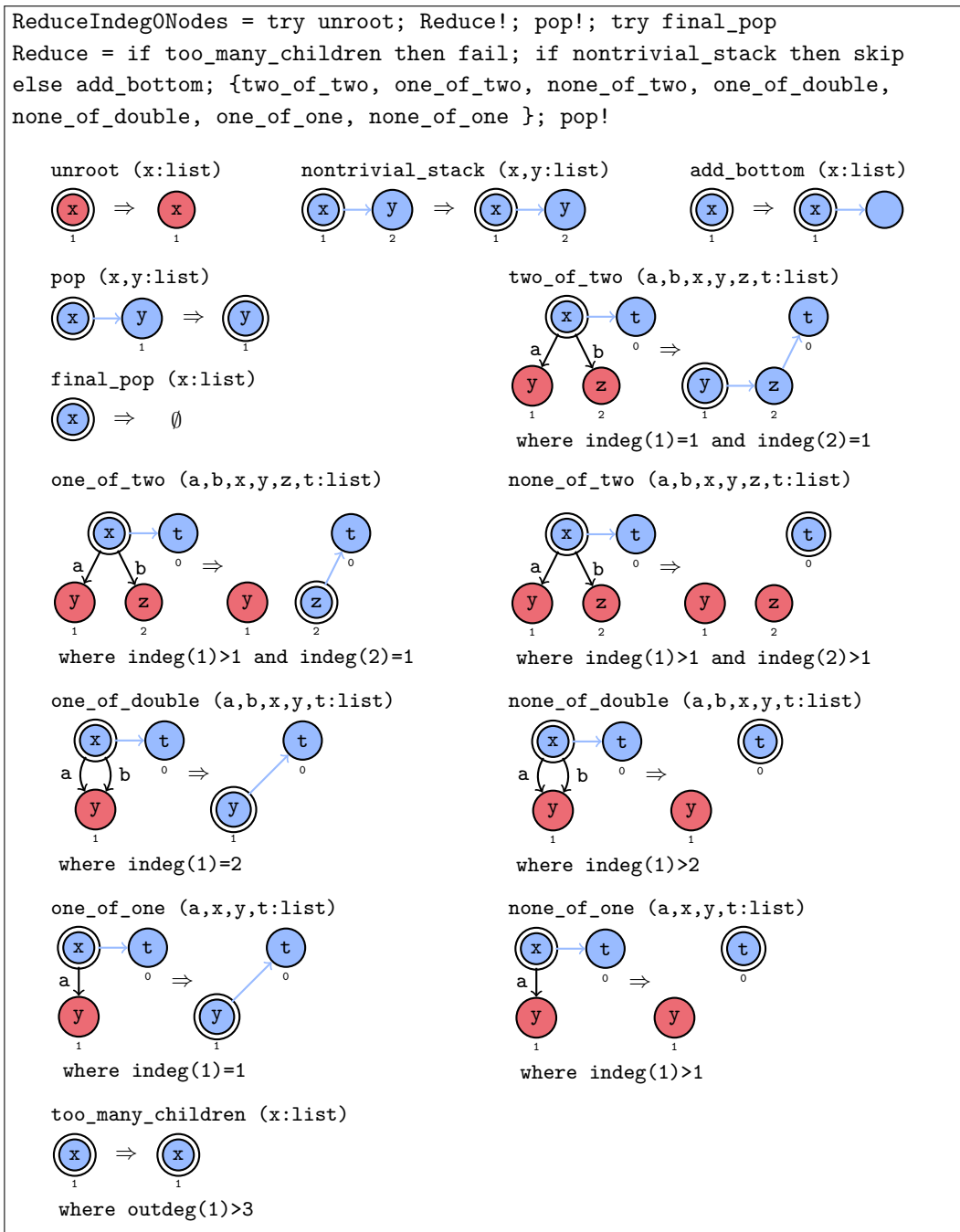


Figure 8 The ReduceIndeg0Nodes procedure.

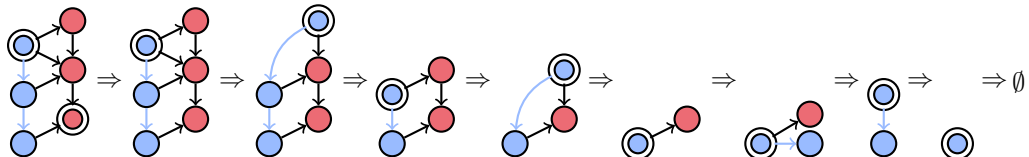


Figure 9 Example execution of ReduceIndeg0Nodes.

never matched by the rule set call rules that would mark them blue. Since there are no rules that delete red nodes (only rules that mark them blue), v_1, v_2, \dots, v_n never get deleted. Thus the output is non-empty. Failure cannot occur since every rule and procedure of `ReduceIndeg0Nodes` is either preceded by `try` or followed by `!`.

Now assume that G minus the blue edges is a DAG but is not binary. Consider a node v of G with no incoming unmarked edges, which exists since G minus the blue edges is a DAG. The aim is to show that, if v has more than two children (excluding blue edges), then the output is non-empty. By Lemma 24, v gets marked blue at some point of the execution. This can only happen in the rule set call rules. Assume v has just been marked blue by one of these rules. We can also assume that v is rooted since, by Lemma 25, every blue node gets deleted at some point, which can only happen in one of the rule set call rules or in `pop`. The case of it happening in `pop` shall be discarded since that would mean v has no children (disregarding blue edges). Back in the execution right after execution of one of the rule set call rules, since `pop!` cannot fail, the loop `Reduce!` enters its next iteration. The procedure tries to apply `too_many_children` to the blue root. If v has more than two children (disregarding blue edges), it succeeds, and the `fail` statement is invoked, terminating the loop `Reduce!`. Since v has children, both `pop` and `final_pop` do not get applied, for the dangling condition is not satisfied. So the output contains v and is therefore non-empty. ◀

4.2 Performance

We will show that our binary DAG recognition program always terminates in linear time, given a connected input graph of bounded degree. We have also included empirical evidence for this.

► **Theorem 13** (Complexity of `is-bin-dag`). *Given a connected input graph of bounded degree, the program `is-bin-dag` terminates in linear time.*

Proof. The Main procedure of `is-bin-dag` contains no loops. `SearchIndeg0Nodes` terminates in linear time by Lemma 20.

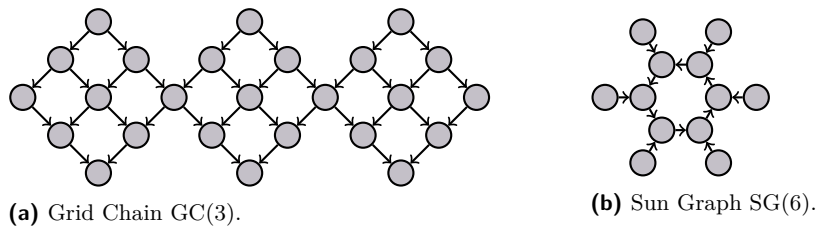
Now consider `ReduceIndeg0Nodes`. By Lemma 23, the procedure terminates. All of its rules are fast, and are hence applied in constant time by Theorem 1 (the input is assumed to have bounded degree, and from the input specification, the fact that `unroot` removes a red root if it is present, and the fact that all the other rules conserve the number of roots, there are at most two roots in the host graph at any given point of the execution). So it is enough to show that each of the constantly many rules gets applied a linear number of times. `unroot` and `final_pop` get applied at most once since they are not inside loops. By the proof of Lemma 23, `add_bottom` gets applied at most twice, and each rule set call rule as well as `pop` at most $|V_G| + 2$ times. `too_many_children` and `nontrivial_stack` can only get reapplied if the rule set call does not fail, which can only happen at most $|V_G| + 2$ times. Hence `ReduceIndeg0Nodes` terminates in linear time.

`nonempty_stack` matches in constant time by Theorem 1 since it is a fast rule. `anything` also matches in constant time since any node is a valid match. ◀

In order to support the linear time complexity of `is-bin-dag`, performance will be measured on two graph classes, one consisting of binary DAGs, and the other of non-DAGs.

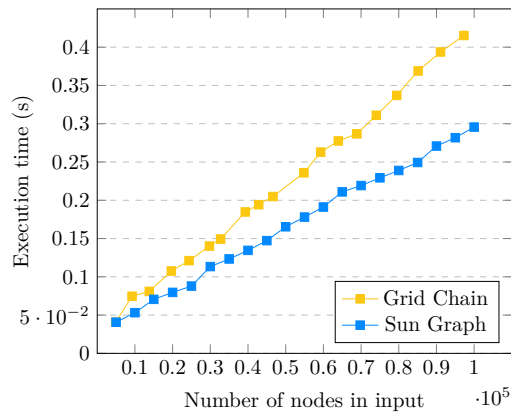
Consider the following class of binary DAGs. For $n \geq 1$, the *grid chain* $GC(n)$ consists of n grids of size $n \times n$, joint by the nodes of indegree and outdegree 1 in order to form a chain. This class was chosen for having an unbounded number of indegree-0 nodes, meaning that the implemented stack is relatively large.

16:12 Linear-Time Graph Algorithms in GP 2



■ **Figure 10** Input Graph Classes.

Now consider the following class of non-DAGs. For $n \geq 3$, the *sun graph* $SG(n)$ consists of a directed cycle of n nodes, each of which has an additional neighbour connected by an incoming edge. The reason for using this class is, in addition to half the nodes having indegree 0, the other half are part of the cycle, and therefore never get deleted by `ReduceIndeg0Nodes`. This causes an unbounded amount of nodes to be left over.



■ **Figure 11** Measured performance of `is-bin-dag`.

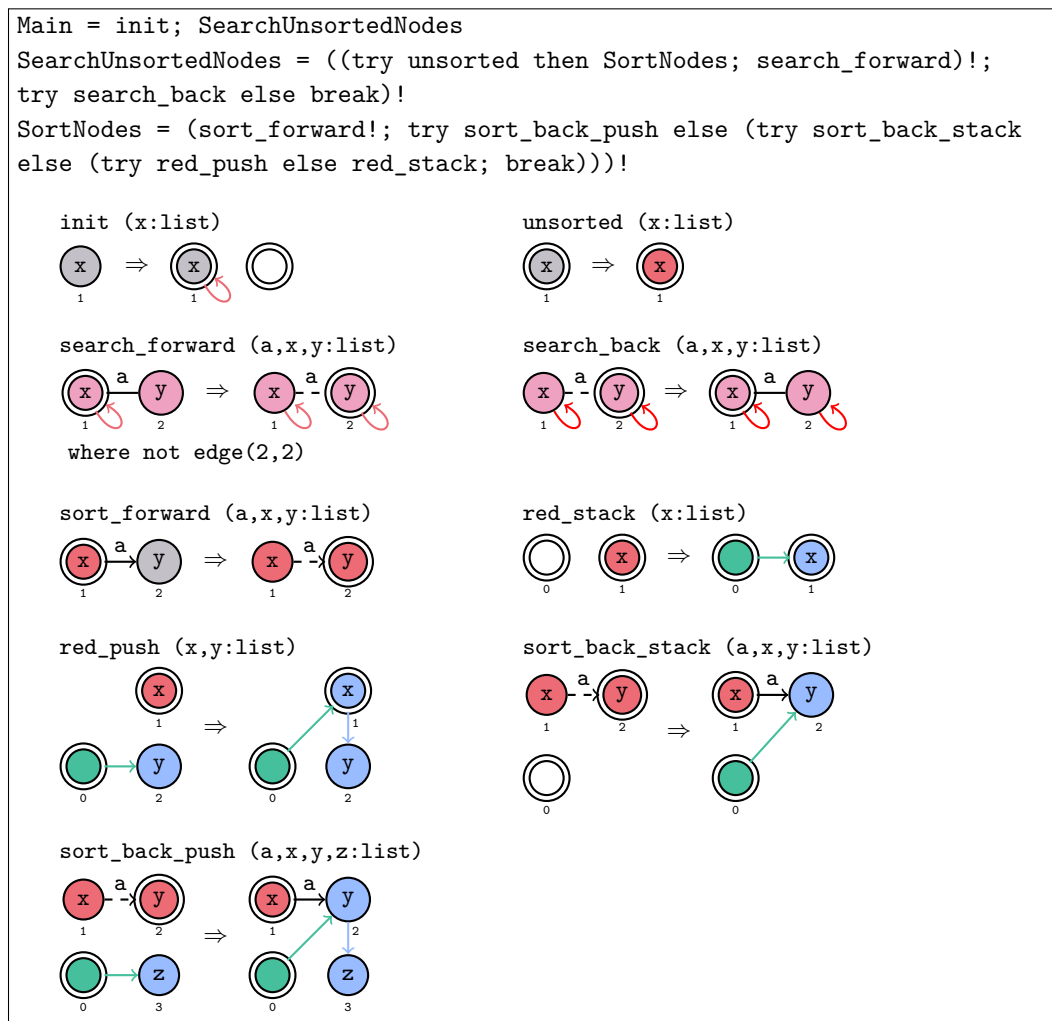
5 Topological Sorting

Given a DAG G , a *topological sorting* is a total order (an antisymmetric, transitive, and connex binary relation) $<$ on V_G , the set of nodes of G , such that for each edge of source u and target v , $u < v$ (*topological property*). Topological sortings cannot exist for graphs containing directed cycles, since there is no way to define a total order on the nodes of a cycle such that the topological property is satisfied. Furthermore, every DAG has a topological sorting.

There are two commonly used linear-time algorithms for finding a topological sorting [20, 19]. One seeks out indegree-0 nodes, adds them to the total order, deletes them, and repeats this process until all nodes have been added to the order. The other traditional algorithm, which is used as the basis for the program `top-sort`, traverses the graph using depth first search (DFS). Upon completion of a node in that DFS, that node is added as the new minimum element of the total order. Note that our DFS will be directed, in the sense that the direction of the edges needs to be respected in order to get a topological sorting in the end. However, this is not enough since that would only visit the nodes reachable from the initially rooted node, which is not necessarily the entire input graph. Hence an operation is needed that efficiently finds an unvisited node once the directed DFS gets stuck.

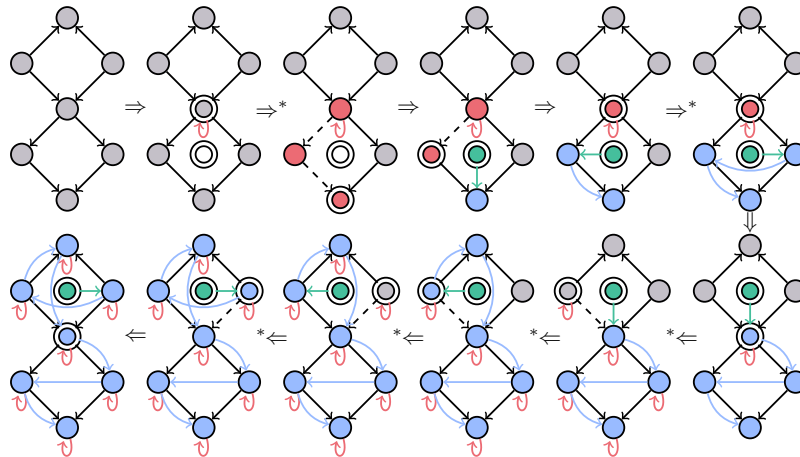
Searching for an unvisited node with a simple rule application will not work because overall it may need to be applied a linear number of times, with single matches requiring linear time. Instead, once the program `top-sort` runs out of unvisited nodes, it uses a second DFS that ignores edge orientation to find a node that has not been sorted yet, and then continues the `SortNodes` DFS on said node. The DFS applications that look for unsorted nodes attach red loops to visited nodes in order to visit any node only once. In this way, the amortized cost of all undirected DFS applications will be linear.

5.1 The Program



■ **Figure 12** The GP2 program `top-sort`.

We give the GP2 implementation of topological sorting in Figure 12 and show its correctness. We have added the restriction that the input graph must be connected since in the current version of GP2, there is no known way to implement a DFS that is linear-time for graphs with an unbounded number of connected components. We have also included an example execution of the program in Figure 13.



■ **Figure 13** Example execution of `top-sort`.

The subgraph induced by the blue edges is a path graph, or linked list, containing all the nodes from the input graph. So the binary relation $<$ on the set of nodes defined by $u < v$ if there is a path of blue edges from u to v is a total order, which is a necessary property for a topological sorting. Similarly to the `SearchIndegONodes` procedure described in Subsection 4.1, the blue nodes and edges implement a stack. However, this time the top of the stack is denoted with a green root pointing towards it with a green edge in order not to interfere with a DFS in `SortNodes`.

The program starts by rooting an input node and endowing it with a red loop, as well as creating an unmarked, unlabelled root that is disconnected from the rest of the graph. This root will point to the top of the stack, and shall hence be called the “pointer”.

`SearchUnsortedNodes` is a DFS implementation that seeks out a node that has not been visited by `SortNodes` yet. Instead of using a red mark to designate a node as visited, it uses a red loop. Since the input is assumed to be a DAG, it has no loops. This leaves the use of marks to the DFS in `SortNodes`. So in order for the forward step to only match unvisited neighbours of the root, a predicate to forbid loops is needed. The “any” mark ensures that colour does not matter. Right before each application of the forward step, `unsorted` tests whether the current root has been visited by `SortNodes` yet, i.e. whether it is grey. At the same time, said root is initialised for `SortNodes` by being marked red.

Next, `SortNodes` is applied. It performs a DFS with directed edges. Similarly to `SearchIndegONodes` from Section 4, it pushes the current root onto the stack during its back step. `sort_back_push` is applied when the stack has at least one element, otherwise `sort_back_stack` creates the stack. The pointer being green represents the stack being non-empty. The break statement is preceded by `try red_push else red_stack`, since when the back step can no longer be applied, the current root is still pushed onto the stack. Again, two rules are needed to cover the cases of the stack being empty or not. Because of the repeated application of the back step, the root ends up where it was at the beginning of `SortNodes`, meaning that the DFS of `SearchUnsortedNodes` can resume undisturbed.

► **Theorem 14** (Correctness of `top-sort`). *The program `top-sort` fulfills the following specification.*

- **Input:** A connected DAG G with no roots whose nodes are all marked grey, and whose edges are unmarked.

- Output: G with additional blue edges that define a topological ordering on V_G . The nodes of G are marked blue and each have a red loop. One of these nodes is rooted. Furthermore, there is an additional unlabelled green root node with an outgoing green edge pointing to a node with no incoming blue edges.

Proof sketch. None of the rules of `try_unsorted` then `SortNodes` modify red looped edges (used by the DFS). Also, after application of `SortNodes`, the red root remains at the same place, and the same edges remain dashed. One can check that `SearchUnsortedNodes` visits every node of its input graph.

`SearchUnsortedNodes` applies `SortNodes` to each of these visited nodes that are marked grey, say v , and implements a stack on $\text{Desc}(v)$ (Definition 27), defining a topological sorting (Lemma 28). Clearly, the subgraph induced by the union of all these descendant graphs is just the output graph. So the concatenation of their topological sortings is a topological sorting of the entire output graph. ◀

The additional constructs in the output graph, apart from the blue edges, are needed for the execution of the program. One could define a linear-time cleanup procedure to remove these constructs. The green root and its outgoing edge can be deleted in constant time, since access to roots is constant. Similarly, the blue rooted node can be unrooted in constant time. A DFS can be used to remove the red loops or unmark all the nodes in linear time.

5.2 Performance

Finally, we show that, given a valid input graph of bounded degree, our topological sorting program will always terminate in linear time.

► **Theorem 15** (Complexity of `top-sort`). *Given a connected DAG of bounded degree with only grey unrooted nodes whose edges are unmarked as an input, the program `top-sort` terminates in linear time.*

Proof sketch. First, let us give an upper bound to the number of applications of each rule. `init` is applied exactly once. Since `init` is the only rule having an unmarked root in its right hand side, and the input has no unmarked roots, `red_stack` and `sort_back_stack` can be matched at most once (in total). `unsorted` and `sort_forward` reduce the number of grey nodes by one. Since all the other rules conserve the number of grey nodes, and the input graph has $|V_G|$ grey nodes, they can be applied at most $|V_G|$ times in total. Similarly, `search_forward` (and `init`) reduce the number of nodes with no red looped edge by one. So they can also only be applied at most $|V_G|$ times in total. `red_push` and `sort_back_push` (as well as `red_stack` and `sort_back_stack`) are the only rules not to conserve the number of blue nodes, and reduce the number of non-blue nodes by exactly one. Since the input graph has no blue nodes, they can be applied at most $|V_G|$ times in total. One can check that `search_back` is applied an at most linear amount of times, since `SortNodes` conserves the number of dashed edges by Lemma 29.

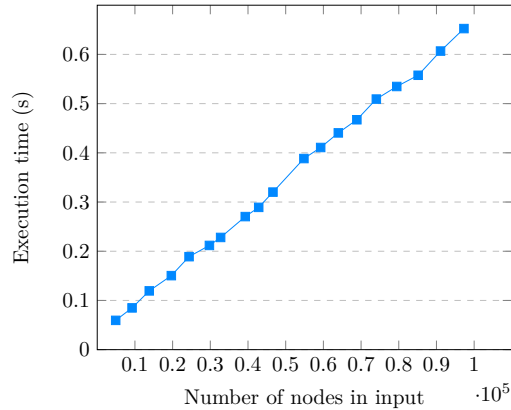
`init` is the only rule to increase the number of roots, specifically by two. All the other rules conserve the number of roots. So since the input graph has no roots, there is a constant number of roots at any point during the execution of `top-sort`.

The only rules that are not fast are `init` due to the lack of roots, and `search_forward` due to the `edge` predicate. So by Theorem 1, all the other rules can be matched in constant time since the input has bounded degree. `init` is matched in constant time since it matches any input node. As for `search_forward`, since the input has bounded degree and the

16:16 Linear-Time Graph Algorithms in GP 2

rules cannot create an unbounded number of edges incident to a single node, the predicate `edge(2,2)` only has to check a constant number of incident edges.

Since each rule is matched a linear number of times in constant time, and the program terminates by Lemma 26, `top-sort` terminates in linear time. ◀



■ **Figure 14** Measured performance of `top-sort` on grid chains.

In order to support the linear time complexity of `top-sort`, we make use of the grid chains from Subsection 4.2. They are DAGs, the type of graph `top-sort` is meant to be used on. Furthermore, they have an unbounded number of indegree-0 nodes. Since indegree-0 nodes are unreachable from any other node, and `SortNodes` can only visit nodes reachable from the red root it is called on, `SortNodes` will have to be applied at least once for each indegree-0 node, i.e. an unbounded number of times. Thus these input graphs can adequately illustrate the linearity of `top-sort`. Figure 14 is a plot of the program timings, demonstrating linear time complexity.

6 Conclusion

The polynomial cost of graph matching is the performance bottleneck for languages based on standard graph transformation rules. GP 2 mitigates this problem by providing rooted rules which under mild conditions can be matched in constant time. We presented rooted GP 2 programs for three graph algorithms: tree recognition, connected binary DAG recognition, and topological sorting. The programs were proved to be correct and to run in linear time on graphs of bounded node degree. The proofs demonstrate that graph transformation rules provide a convenient and intuitive abstraction level for formal reasoning on graph programs. We also gave empirical evidence for the linear run time of the programs, by presenting benchmark results for graphs of up to 100,000 nodes in various graph classes. For DAG recognition and topological sorting, the linear behaviour was achieved by implementing depth-first search strategies based on an encoding of stacks in graphs.

In future work, we intend to investigate for more graph algorithms whether and under what conditions their time complexity in conventional programming languages can be reached in GP 2. The more involved the data structures of those algorithms are, the more challenging will be the implementation task. This is because in GP 2, the internal graph data structure is (intentionally) hidden from the programmer and hence any data structures used by an algorithm need to be encoded in host graphs. A simple example for this is the encoding of stacks as linked lists in the programs for DAG recognition and topological sorting.

Additional future work is the automated refinement of programs, adding root nodes in order to improve matching performance. It is highly non-obvious how to do this in general, or what refinement tactics could be used. It is possible that DFS can provide a framework for combining procedures in an efficient way.

The three programs in this paper and also the 2-colouring program of [6] need host graphs of bounded node degree in order to run in linear time. A topic for future work is therefore to find a mechanism that allows to overcome this restriction. Clearly, such a mechanism will require to modify GP 2 and its implementation.

References

- 1 Aditya Agrawal, Gabor Karsai, Sandeep Neema, Feng Shi, and Attila Vizhanyo. The design of a language for model transformations. *Software and System Modeling*, 5(3):261–288, 2006. doi:10.1007/s10270-006-0027-7.
- 2 Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.
- 3 Thorsten Arendt, Enrico Biermann, Stefan Jurack, Christian Krause, and Gabriele Taentzer. Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations. In *Model Driven Engineering Languages and Systems (MODELS 2010)*, volume 6394 of *Lecture Notes in Computer Science*, pages 121–135. Springer, 2010. doi:10.1007/978-3-642-16145-2_9.
- 4 Christopher Bak. *GP 2: Efficient Implementation of a Graph Programming Language*. PhD thesis, Department of Computer Science, University of York, 2015. URL: <http://etheses.whiterose.ac.uk/12586/>.
- 5 Christopher Bak and Detlef Plump. Rooted Graph Programs. In *Proc. International Workshop on Graph Based Tools (GraBaTs 2012)*, volume 54 of *Electronic Communications of the EASST*, 2012. doi:10.14279/tuj.eceasst.54.780.
- 6 Christopher Bak and Detlef Plump. Compiling Graph Programs to C. In *Proc. International Conference on Graph Transformation (ICGT 2016)*, volume 9761 of *LNCS*, pages 102–117. Springer, 2016. doi:10.1007/978-3-319-40530-8_7.
- 7 Heiko Dörr. *Efficient Graph Rewriting and its Implementation*, volume 922 of *Lecture Notes in Computer Science*. Springer, 1995. doi:10.1007/BFb0031909.
- 8 Hartmut Ehrig, Claudia Ermel, Ulrike Golas, and Frank Hermann. *Graph and Model Transformation*. Monographs in Theoretical Computer Science. Springer, 2015. doi:10.1007/978-3-662-47980-3.
- 9 Maribel Fernández, Hélène Kirchner, Ian Mackie, and Bruno Pinaud. Visual Modelling of Complex Systems: Towards an Abstract Machine for PORGY. In *Proc. Computability in Europe (CiE 2014)*, volume 8493 of *Lecture Notes in Computer Science*, pages 183–193. Springer, 2014. doi:10.1007/978-3-319-08019-2_19.
- 10 Amir Hossein Ghamarian, Maarten de Mol, Arend Rensink, Eduardo Zambon, and Maria Zimakova. Modelling and analysis using GROOVE. *International Journal on Software Tools for Technology Transfer*, 14(1):15–40, 2012. doi:10.1007/s10009-011-0186-x.
- 11 Annegret Habel and Detlef Plump. Relabelling in Graph Transformation. In *Proc. International Conference on Graph Transformation (ICGT 2002)*, volume 2505 of *Lecture Notes in Computer Science*, pages 135–147. Springer, 2002. doi:10.1007/3-540-45832-8_12.
- 12 Ivaylo Hristakiev and Detlef Plump. Checking Graph Programs for Confluence. In *Software Technologies: Applications and Foundations – STAF 2017 Collocated Workshops, Revised Selected Papers*, volume 10748 of *Lecture Notes in Computer Science*, pages 92–108. Springer, 2018. doi:10.1007/978-3-319-74730-9_8.
- 13 Edgar Jakumeit, Sebastian Buchwald, and Moritz Kroll. GrGen.NET - the expressive, convenient and fast graph rewrite system. *International Journal on Software Tools for Technology Transfer*, 12(3–4):263–271, 2010. doi:10.1007/s10009-010-0148-8.

- 14 Detlef Plump. The Design of GP 2. In *Proc. Workshop on Reduction Strategies in Rewriting and Programming (WRS 2011)*, volume 82 of *Electronic Proceedings in Theoretical Computer Science*, pages 1–16, 2012. doi:10.4204/EPTCS.82.1.
- 15 Detlef Plump. From Imperative to Rule-based Graph Programs. *Journal of Logical and Algebraic Methods in Programming*, 88:154–173, 2017. doi:10.1016/j.jlamp.2016.12.001.
- 16 Christopher M. Poskitt and Detlef Plump. Hoare-style verification of graph programs. *Fundamenta Informaticae*, 118(1-2):135–175, 2012. doi:10.3233/FI-2012-708.
- 17 Christopher M. Poskitt and Detlef Plump. Verifying Monadic Second-Order Properties of Graph Programs. In *Proc. International Conference on Graph Transformation (ICGT 2014)*, volume 8571 of *Lecture Notes in Computer Science*, pages 33–48. Springer, 2014. doi:10.1007/978-3-319-09108-2_3.
- 18 Olga Runge, Claudia Ermel, and Gabriele Taentzer. AGG 2.0 — new features for specifying and analyzing algebraic graph transformations. In *Proc. Applications of Graph Transformations with Industrial Relevance (AGTIVE 2011)*, volume 7233 of *Lecture Notes in Computer Science*, pages 81–88. Springer, 2012. doi:10.1007/978-3-642-34176-2_8.
- 19 Robert Sedgewick. *Algorithms in C. Part 5: Graph Algorithms*. Addison-Wesley, third edition, 2002.
- 20 Steven Skiena. *The Algorithm Design Manual*. Springer, second edition, 2008. doi:10.1007/978-1-84800-070-4.

A Appendix: Proofs

This appendix consists of lemmata and proofs omitted from the main sections.

A.1 Tree Recognition Lemmata

By *rooted input graph*, we mean an arbitrary labelled GP 2 input graph with every node coloured grey, exactly one *root* node, and no additional marks. That is, a valid input graph after `init` has been applied. By *rooted input tree*, we mean an rooted input graph that is a tree. In this appendix, we give the proofs of the lemmata needed to support Proposition 6 and Theorem 7 from Section 3. Note that an equivalent characterisation of a tree is a non-empty connected graph without undirected cycles such that every node has at most one incoming edge.

► **Lemma 16.** *If G is a tree and $G \Rightarrow_{Reduce}^* H$, then H is a tree. If G is not a tree and $G \Rightarrow_{Reduce} H$, then H is not a tree.*

Proof. Clearly, the application of `push` preserves structure. Suppose G is a tree. `prune` is applicable if and only the second node is matched against a leaf node, due to the dangling condition. Upon application, the leaf node and its incoming edge is removed. Clearly the result graph is still a tree. If G is not a tree and `prune` is applicable, then we can see the properties of not being a tree are preserved. That is, if G is not connected, H is certainly not connected. If G had parallel edges, due to the dangling condition, they must exist in $G \setminus g(L)$, so H has parallel edges. Similarly, cycles are preserved. Finally, if G had a node with incoming degree greater than one, then H must too, since the node in G that is deleted in H had incoming degree one, and the degree of all other nodes is preserved. So, we have shown `Reduce` is structure preserving, and then by induction, so is `Reduce!`. ◀

► **Lemma 17.** *If G is a rooted input graph and $G \Rightarrow_{Reduce}^* H$, then H has exactly one root node. Moreover, there is no derivation sequence that derives the empty graph.*

Proof. In each application of `prune` or `push`, the number of root nodes is invariant since the LHS of each rule must be matched against a root node in the host graph, so the other non-roots can only be matched against non-roots, and so the result holds by induction. To see that the empty graph cannot be derived, notice that each derivation reduces $\#G$ by at most one, and no rules are applicable when $\#G = 1$. ◀

► **Lemma 18.** *If G is a rooted input graph and $G \Rightarrow_{Reduce}^* H$. Then, every blue node in H either has a blue child or a root-node child.*

Proof. As there are no blue nodes, G satisfies this. We now proceed by induction. Suppose $G \Rightarrow_{Reduce}^* H \Rightarrow_{Reduce} H'$ where H satisfies the condition. If `prune` is applicable, we introduce no new blue nodes. Additionally, any blue parents of the node 1 are preserved. Finally, if `push` is applied, then the new blue node has a root-node child, and the blue nodes in $H' \setminus h(R)$ have the same children. So H' satisfies the condition. ◀

► **Corollary 19.** *Let G be a rooted input tree and $G \Rightarrow_{Reduce}^* H$. Then the root-node in H has no blue children.*

Proof. By Lemma 17, H has exactly one root node, and by Lemma 18, all chains of blue nodes terminate with a root-node. If said root-node were to have a blue child, then we would have a cycle, which contradicts that H is a tree (Lemma 16). ◀

A.2 Binary DAG Recognition Lemmata

In this appendix, we give the proofs of the lemmata needed to support Propositions 11, 12, and Theorem 13 from Section 4.

► **Lemma 20** (Complexity and Partial Correctness of `SearchIndeg0Nodes`). *Given a connected input graph G with grey unrooted nodes and unmarked edges, `SearchIndeg0Nodes` terminates, and the subgraph H induced by the edges that have been dashed during the execution is a spanning tree. Furthermore, if G has bounded degree, the procedure terminates in linear time.*

Proof sketch. Similar to the proofs in those given by Bak and Plump [5] [4]. ◀

► **Lemma 21.** *Given a non-empty connected input graph G with grey unrooted nodes and unmarked edges, at any point of the execution of `SearchIndeg0Nodes`, there is at most one red root.*

Proof sketch. `init` introduces a red root, and is only applied once and in the beginning. The other rules that do not preserve red roots are `i0_push`, `i0_stack` and `i0_back_blue`. If either `i0_push` or `i0_stack` are applied, the red root vanishes. Subsequently, `i0_back_red` cannot be applied. If `i0_back_blue` then gets applied the red root is reintroduced, conserving the existence of a red root within the iteration of the loop. If `i0_back_blue` does not get applied, the `break` statement is invoked and the procedure terminates. ◀

► **Lemma 22.** *Given a non-empty connected input graph G with grey unrooted nodes and unmarked edges, `SearchIndeg0Nodes` outputs G where all the indegree-0 nodes (and only those) are marked blue and connected with blue edges forming a path graph. The blue node with no incoming blue edge is rooted.*

Proof sketch. If G has no indegree-0 nodes, then the lemma is trivially satisfied. So assume G has at least one.

16:20 Linear-Time Graph Algorithms in GP 2

By Lemma 20, `SearchIndeg0Nodes` visits all nodes. Every node in the output graph is marked red or blue. Blue nodes can only come from indegree-0 nodes matched by `i0_push` or `i0_back_blue`.

Since the right hand side of each rule only contains red and blue nodes, every node is marked either red or blue. The only rules that introduce a blue mark are `i0_push` and `i0_back_blue`, and they turn a red root into a blue root. These rules only get applied if the indegree of said red node is 0. Furthermore, the only edges introduced by `SearchIndeg0Nodes` are blue edges between two blue nodes (in `i0_push`), hence the indegree of a red node is the same as its indegree in the input graph. So only indegree-0 nodes are marked blue. Furthermore, since `SearchIndeg0Nodes` visits, i.e. roots every node of the input graph at some point, all indegree-0 nodes are marked blue, and all non-indegree-0 nodes red.

All rules apart from `i0_push` and `i0_stack` preserve the structure of the subgraph consisting of blue nodes and edges. `i0_stack` only is applied only if `i0_push` is not applicable. But the left hand side of `i0_push` contains a blue root, which can only be created by itself or `i0_stack`. So `i0_push` cannot be applied until `i0_stack` is applied. Since G cannot consist of only indegree-0 nodes (which would mean G is disconnected), `i0_push` can always be matched if the red root has indegree 0. If the red root does not have indegree 0, `i0_stack` cannot be matched either. So the only way for these two rules to match is for `i0_stack` to be matched first and only once, followed by `i0_push` being matched any number of times. Thus, a blue root is created, and then, repeatedly, a new blue node gets connected to the blue root with an outgoing blue edge, while the root moves to the newly added blue node. This construction results in the blue nodes and edges forming a path graph where the node with no incoming edges is a root. ◀

► **Lemma 23** (Termination of `ReduceIndeg0Nodes`). *Let G be a connected graph with red non-indegree-0 nodes containing at most one root, and blue indegree-0 nodes that are connected with blue edges forming a path graph. The blue node with no incoming blue edges is a root.*

Given a G as an input, `ReduceIndeg0Nodes` terminates.

Proof sketch. `pop` can only be applied a finite number of times since it reduces the number of nodes in the host graph. So `pop!` terminates. One can check that during the execution of `ReduceIndeg0Nodes`, `add_bottom` gets applied at most twice. The rules in the rule set `call` and `pop` reduce the number of nodes in the host graph by exactly one. So by the claim, they can be applied at most $|V_G| + 2$ times each. So at some point in the loop, they will no longer be applicable. Neither will `add_bottom` since it can only be applied twice. So `Reduce!` terminates. ◀

► **Lemma 24.** *Given an input graph G as described in Lemma 23, every node that has no incoming unmarked edges (called quasi-indegree-0 node) in some host graph of the execution of `ReduceIndeg0Nodes` gets marked blue.*

Proof sketch. Indeed, the input graph has all quasi-indegree-0 nodes marked blue already. The only rules deleting edges are those from the rule set `call` (`pop` and `final_pop` cannot delete unmarked edges incident to the node they delete because the dangling condition needs to be satisfied for them to match). So these are the only rules that can create new quasi-indegree-0 nodes. If one of said nodes has indegree 0, it gets detected by the condition of a rule and marked blue. These rules cover each case of how many children their quasi-indegree-0 parent can have in a binary DAG, namely one, one with two parallel edges, and two. The case of no children is covered by `pop` afterwards. They also cover all cases of how many of these children are quasi-indegree-0. So at each execution step, the newly created quasi-indegree-0 nodes get marked blue, proving this lemma. ◀

► **Lemma 25.** *Given an input graph G as described in Lemma 23, every node that is marked blue during execution of `ReduceIndegONodes` is not present in the output.*

Proof sketch. Nodes can only be marked blue if an already existing blue node is matched. So it is enough to show that, at some point of the execution, there will be no blue nodes. There are three potential ways to exit the loop `Reduce!`. The first is through the `fail` statement after matching `too_many_children`. This will never happen since the input minus the blue edges is binary, and every rule conserves the blue root having exactly one outgoing blue edge. The second way is for `add_bottom` to fail. This can only happen when there is no blue root. The only rule deleting a blue root is `final_pop`, which is only called after termination of `Reduce!`. Since furthermore, the input is assumed to have a blue root, and every other rule conserves the existence of a blue root, `add_bottom` is always applicable. The third and final way to exit the loop is when none of the rules in the rule set call are applicable. The blue root not having an element below it in the stack cannot be a reason for that, since in that case, `add_bottom` would have been applied. So the current blue root v does not have red neighbours. Since `pop!` has been applied in the previous iteration of `Reduce!`, v was the only blue node in the previous iteration, otherwise it would have been popped. Hence in the current iteration, `add_bottom` was applied, and so the only blue nodes are v and the node created by `add_bottom`, say w . By Lemma 23, `Reduce!` terminates, so this always happens for the given input. As established, v has no children. Neither does w since it was created by `add_bottom` and there is no rule with edges incident to red nodes in its right hand side. Thus `pop` deletes v , then `final_pop` deletes w , causing all previously blue marked nodes to be deleted. ◀

A.3 Topological Sorting Lemmata

In this appendix, we give the proofs of the lemmata needed to support Theorems 14 and 15 from Section 5.

► **Lemma 26** (Termination of `top-sort`). *Given a connected DAG G with no roots, grey nodes, and unmarked edges as an input, `top-sort` terminates.*

Proof sketch. `sort_forward!` terminates since in each iteration, the number of grey nodes decreases.

For the termination of `SortNodes`, consider the following lexicographical ordering $>$. $H_1 > H_2$ if one of the following three statements are satisfied. H_1 has more grey nodes than H_2 , or they have the same number of grey nodes but H_1 has more dashed edges, or they have the same number of grey nodes and dashed edges but H_1 has more red nodes. Let H_1 be the input of an arbitrary iteration of `SortNodes`, and H_2 its output. If `sort_forward` is applied any number of times, $H_1 > H_2$ since the number of grey nodes are reduced. Otherwise, if either `sort_back_push` or `sort_back_stack` is applied, $H_1 > H_2$ since the number of grey nodes is conserved and the number of dashed edges decreases in both rules. Otherwise, either `red_push` or `red_stack` have to be applied, which conserve the number of grey nodes and dashed edges, but decreases the number of red nodes. So in any case, $H_1 > H_2$. For a given graph H_1 consider how many graphs H_2 satisfy $H_1 > H_2$. By definition of $>$, H_1 gives a (finite) upper bound on the number of grey nodes, dashed edges, and red nodes. Hence there are only finitely many possible H_2 s. Since `sort_forward!` terminates, and each iteration of the loop reduces the host graph with respect to $<$, `SortNodes` terminates.

Consider `(try unsorted then SortNodes; search_forward)!`. If `search_forward` cannot be applied, the loop terminates. It is the only rule in this loop that increases the number of looped edges in the graph. Due to its predicate, it can only add looped edge to a

16:22 Linear-Time Graph Algorithms in GP 2

node if it does not already have one. Furthermore, no rule decreases the number of looped edges. So for an arbitrary input graph H for the loop, at most $|V_H|$ looped edges can be added before `search_forward` fails. Hence the loop terminates.

Finally, consider the loop that `SearchUnsortedNodes` consists of. Furthermore, consider the lexicographic ordering $>$ defined by $H_1 > H_2$ if H_2 has more nodes with looped edges than H_1 , or they have the same number of nodes with looped edges but H_2 has less dashed edges than H_1 . By an argument similar to that made by Bak for termination of DFS [4], `SearchUnsortedNodes` terminates. ◀

For the correctness of `SortNodes`, the following concepts need to be defined. In a graph G , a *directed path* from a node v to a node w is a sequence of distinct nodes v_1, v_2, \dots, v_n such that $v_1 = v$ and $v_n = w$, and for each i where $1 \leq i \leq n - 1$, there is an edge of source v_i and of target v_{i+1} . A directed path from v to w is called *grey-noded* if all the nodes it consists of, except possibly v , are marked grey.

► **Definition 27** (Descendants). *Given a node v in a DAG G , let its descendants $Desc_G(v)$ be defined as the subgraph of G induced by the set*

$$\{w \in V_G \mid \text{there is a directed grey-noded path from } v \text{ to } w\} \cup \{v\}.$$

► **Lemma 28** (Correctness of `SortNodes`). *Assume the input graph of `top-sort` has no blue edges. Let G be a connected DAG with a single red root v , where the nodes of $Desc_G(v)$ are unrooted. Furthermore, let G have an additional root that is either unmarked and disconnected, or green and connected to the rest of the graph with an outgoing green edge. Let H be the output of `SortNodes` applied on G . Consider the binary relation $<$ on nodes of $Desc_H(v)$ defined by $u < w$ if there is a directed path from u to w , such that all of the involved edges are blue. Then $<$ defines a topological sorting on $Desc_H(v)$ minus the blue edges.*

Proof sketch. Since the input graph of `top-sort` has no blue edges, any that are present in the host graph were created by rules. Whenever these rules create blue edges, they mark the incident nodes blue. No rule removes a blue mark, so the subgraph of the host graph induced by the blue edges always exclusively consists of blue nodes. Furthermore, every time a node gets marked blue, the green root points towards it. And when a new blue edge gets created, the target node must also have the green root pointing towards it, and the source node must be a red root. So the procedure only adds a blue edge from a non-blue to the node that has most recently been marked blue. From this construction, we can infer that the graph induced by the blue edges is a path graph. Furthermore, no blue looped edges are introduced. So there can be no path from a node u to a node w and vice versa. Hence if $u < w$ and $w < u$, u and w must be equal by definition of $<$, i.e. \leq is antisymmetric.

From the definition of $<$, it is clear that transitivity holds due to path concatenation resulting in paths.

One can show that `SortNodes` turns every node of $Desc_G(v)$ into a red root. Furthermore, all the red roots become blue nodes incident to blue edges. So $<$ is connex.

To show that the topological property holds, consider two nodes u and w of $Desc_H(v)$, both of which being distinct from v (v itself will be handled later). So by definition, there is a path of non-blue edges from v to u , and one from v to w . We can assume without loss of generality that u becomes a red root before w . If there is no edge between u and w , the topological property imposes no constraint on said pair of nodes. If there is an edge from u to w , `sort_forward` gets applied again, dashing said edge and turning w into a red root. Hence later in the execution, w gets pushed before u , ensuring that the topological property

is satisfied. If there is an edge from w to u , there can be no non-blue path from u to w since the input is a DAG. Hence u will be pushed before w , satisfying the topological property again. As for v , any condition involving it must have it as the source node by definition of $\text{Desc}_H(v)$. Since v is pushed last, the topological property is satisfied. ◀

► **Lemma 29.** *Given an input G as described in Lemma 28, the output of SortNodes has the same dashed edges, and the red root in the same place as G .*

Proof sketch. Let v be the red root of G . During the execution of SortNodes , there is always a path of dashed edges from v to the current red root, since `sort_forward` is the only rule of SortNodes with dashed edges in its right hand side and generates a path graph of red nodes and dashed edges, and since `sort_back_stack` and `sort_back_push` only remove the latest node from that path graph. The only way for their encompassing loop to end is for both of these rules not to be applicable. By the previous argument, this means that there are no dashed edges in said path graph left, and v is the red root when SortNodes terminates. ◀

Hybridisation of Institutions in HETS

Mihai Codescu 

University of Bremen, Collaborative Research Center EASE, Bremen, Germany
Institute of Mathematics “Simion Stoilow” of the Romanian Academy,
Research Group of the project PED-0494, Bucharest, Romania
codescu@uni-bremen.de

Abstract

We present a tool for the specification and verification of reconfigurable systems. The foundation of the tool is provided by a generic method, called hybridisation of institutions, of extending an arbitrary base institution with features characteristic to hybrid logic, both at the syntactic and the semantic level. Automated proof support for hybridised institutions is obtained via a generic lifting of encodings to first-order logic from the base institution to the hybridised institution. We describe how hybridisation and lifting of encodings to first-order logic are implemented in an extension of the Heterogeneous Tool Set in their full generality. We illustrate the formalism thus obtained with the specification and verification of an autonomous car driving system for highways.

2012 ACM Subject Classification Software and its engineering → Specification languages; Theory of computation → Algebraic semantics; Theory of computation → Logic and verification

Keywords and phrases hybrid logics, formal verification, institutions, reconfigurable systems

Digital Object Identifier 10.4230/LIPIcs.CALCO.2019.17

Category Tool Paper

Funding This work was partially supported by the German Research Foundation DFG, as part of Collaborative Research Center (Sonderforschungsbereich) 1320 “EASE - Everyday Activity Science and Engineering”, University of Bremen (<http://www.ease-crc.org/>) and by a grant of the Romanian National Authority for Scientific Research and Innovation, CNCS/CCCDI - UEFISCDI, project number PN-III-P2-2.1-PED-2016-0494, within PNCDI III.

Acknowledgements Răzvan Diaconescu had major contributions to the implementation in HETS of his generic method of hybridisation of institutions. I wish to thank Till Mossakowski, Fabian Neuhaus and Ionuț Țuțu for valuable feedback on language design and implementation issues.

1 Introduction

A *reconfigurable system* is one with different modes of operation, called *configurations*, and with the ability to commute between them during its execution along transitions between these modes, called *reconfigurations*. Such systems appear naturally in many domains, including automobile industry, robotics and medical devices. An overview can be found in [9]. We present H [3], a tool for the formal specification and verification of reconfigurable systems that supports their correct and efficient development.

The mathematical foundation underlying the tool is provided by a generic construction on institutions [8], called *hybridisation*, explained briefly in Sec. 2. It has the modalisation of institutions [7] as its source; part of that work was extended to hybrid logics in [10] in a simple form, and it took a rather complete shape in [5]. Hybridisation is done using a two-layered approach: the base layer represents a specific logic for expressing requirements at the configuration (static) level, in other words at the data level. This layer is treated abstractly as an institution that can be instantiated to concrete logical formalisms that are most adequate for the specification of the data part of particular problems. On top of this base layer the characteristic syntactic and semantics features of hybrid logic are



© Mihai Codescu;

licensed under Creative Commons License CC-BY

8th Conference on Algebra and Coalgebra in Computer Science (CALCO 2019).

Editors: Markus Roggenbach and Ana Sokolova; Article No. 17; pp. 17:1–17:10

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

developed, notably a flexible choice of *quantifications* on nominals and/or symbols from the base institution and various *semantic constraints* on the accessibility relations and on the interpretation of symbols in possible worlds of a Kripke model of the hybridised institution (also see [5]). While the base layer deals with the data level, the upper layer deals with the dynamics of the configurations. The choice of hybrid logic for expressing the latter comes naturally as it is a prominent kind of modal logic that provides adequate syntactic capabilities – names of possible worlds, formulas that hold at named states.

The specification part of our tool is complemented by a verification part, whose foundation is given by a general encoding of hybridised institutions into first-order logic. This encoding follows the two-layered structure of the hybridised institutions. If a translation of the logic used at the base level to first-order logic already exists, it is *lifted* to a translation of the hybridised institution to first-order logic via a generic construction, introduced in [6]. At this level, the semantic constraints give rise to first-order formulas. As a result, we obtain a *verification-by-translation* method where a problem in a hybridised institution is translated to first-order logic and solved there using automated first-order theorem provers.

The H tool was implemented as an extension of the Heterogeneous Tool Set (HETS) [13], a tool for the heterogeneous multi-logic specification and modeling of software systems and for ontology development. In all these fields, there is a large number of logics and languages in use, each better suited for a different task or providing better support for a different aspect of a complex system. Instead of trying to integrate the features of all these logics into a single formalism, the paradigm of heterogeneous multi-logic specification is to integrate all logics by means of a so-called Grothendieck construction over a graph of logics and their translations [12, 4]. Thus, for each logic we can make use of its dedicated syntax(es) and proof tools. The specifier has the freedom to choose the logic that suits best the problem to be solved, offers best tool support and is most familiar with. HETS provides an implementation of this paradigm, and also supports the verification-by-translation method. HETS has been designed as a flexible tool: adding a new logic or a new logic translation can be done by instantiating a class and adding the new instance to the list of known logics and translations. First-order logic and several state-of-the-art automated provers for it are already supported by HETS.

A HETS implementation of the hybridisation method was presented in [14]. It was realized in two directions: an implementation of the hybridisation of an extension of CASL logic with rigid symbols (enabling *user-defined sharing*, when only the symbols explicitly marked as rigid are subject to semantic constraints in the models of the hybridised institution), together with a translation from this hybridisation to first-order logic, and a generic construction, similar to a Grothendieck one, that appears as a single logic in HETS, used to hybridise a number of HETS logics. No translation from this logic to first-order logic is available, and no choice can be made on the kind of quantification and the semantic constraints that a hybridised institution should have. In contrast, our implementation supports all variations and is fully generic: the parameters of the hybridisation method can be specified in a declarative way and new instances of the main `Logic` class of HETS are generated for each new definition of a hybridised institution. Generating different institutions for different hybridisations is crucial for defining comorphisms from them to first-order logic, which is also implemented in our tool as a generic method, thus enabling proof support for each newly added hybridised institution.

2 Institutions and their hybridisation

Institutions [8] provide a model-theoretic formalization of the concept of logical system. The basic components of an institution are: a notion of *signature*, defining the non-logical symbols in the language, a notion of *logical sentence* over a signature, a notion of *model* giving the

interpretation of the symbols in a signature in some semantic domain and a *satisfaction relation* between the models and the sentences of a signature. This is complemented by a dynamic view on the language: instead of working over an arbitrary but fixed (and implicit) signature, different signatures are related by *signature morphisms*, which induce *translations of sentences* and *reduction of models*. These must be consistent with one another, which means that no change of notation induced by a signature morphism can alter the satisfaction of sentences. This is expressed formally by the so-called *satisfaction condition*. Institutions are a formalization of the above using category theory, thus achieving a high level of abstraction and not making unnecessary assumptions about the components of a logical system.

The hybridisation method [5] was introduced at this abstract level. Given an arbitrary base institution \mathcal{I} as a first parameter, it constructs a hybridised institution \mathcal{HI} whose signatures extend the signatures of \mathcal{I} with nominals (for reconfigurable systems, these correspond to names of configurations) and modalities (names of events causing reconfigurations). Signature morphisms in \mathcal{I} pair signature morphisms in \mathcal{I} with mappings of nominals and of modalities. For a signature in \mathcal{I} , the sentences can be \mathcal{I} -sentences over the base signature, nominals, modal box- and diamond-sentences over modalities, *retrieve* sentences (meant to hold at a given state), combinations of sentences using Boolean connectors, or quantified sentences. The latter sentences depend on a class \mathcal{D} of signature morphisms of \mathcal{HI} that defines the kind of variables that can be quantified, nominals and/or symbols from \mathcal{I} , and forms the second parameter of hybridisation. Models of a \mathcal{HI} signature are Kripke structures such that each possible world is assigned a \mathcal{I} -model of the base signature, each nominal is interpreted as one of the possible worlds and each modality as an accessibility relation between these worlds. The third parameter of hybridisation is a set of logic-specific semantic constraints on the accessibility relations and on the interpretation of symbols in possible worlds. For example, the accessibility relation may be reflexive and transitive, as in the modal logic **S4**, or the interpretation of all symbols of a certain kind may be the same in all possible worlds.

Institution comorphisms [11] capture the intuition that an institution is included or encoded into another one. A comorphism from an institution \mathcal{I}_1 to an institution \mathcal{I}_2 maps \mathcal{I}_1 -signatures to \mathcal{I}_2 -signatures along a functor Φ , Σ -sentences in \mathcal{I}_1 to $\Phi(\Sigma)$ -sentences in \mathcal{I}_2 for a \mathcal{I}_1 -signature Σ and \mathcal{I}_2 -models of $\Phi(\Sigma)$ to \mathcal{I}_1 -models of Σ . Again, a satisfaction condition must hold, stating that satisfaction of sentences is not altered by change of logic. Sometimes the cost of encoding an institution \mathcal{I}_1 into another one \mathcal{I}_2 is that \mathcal{I}_1 -signatures are mapped to \mathcal{I}_2 -theories, i.e. not just signatures, but also a set of sentences over them. These theories grow in size with the number of symbols in the original signature.

Given an institution comorphism from an institution \mathcal{I} to the institution of multi-sorted first-order logic FOL^{ms} , [6] introduces a generic method of lifting it to a comorphism from a hybridisation \mathcal{HI} of \mathcal{I} to FOL^{ms} . A hybrid signature Δ get translated to a FOL^{ms} -theory (Σ, E) as follows: first the base signature is translated along the base comorphism, and we obtain a first-order theory. This theory is extended with a new sort for states, its predicates and function symbols get a new argument of sort state, and the sentences of the theory are universally quantified over a variable of sort state that is introduced in all predications and all terms. Domain predicates are introduced for each sort and state, giving the interpretation of that sort in each world. Nominals are constants of sort state and modalities are predicates on states. Moreover, semantic constraints get translated to sentences over this extended signature. The reduction of a (Σ, E) -model to a Δ -model is done by taking as the set of worlds the interpretation of the sort for states, and by keeping the interpretation of nominals and modalities as in the first-order model. The local models are obtained for each world w by taking the reduct along the base comorphism of the first-order model obtained by interpreting

each sort as its domain in w and each function/predicate symbol as the restriction of its interpretation in (Σ, E) when the extra state argument is always w . Sentence translation is done by adding an universal quantification on a variable w of sort state and then inductively on the structure of the formula, with the base cases of nominals i being translated to $i = w$ and base formulas e being first translated along the base comorphism and then adding w in the resulting first-order sentences as the extra argument of sort state. Details of the interesting cases of box- and diamond formulas and quantification can be found in [6].

3 Hybridisation in HETS

The parameters of the generic hybridisation method are:

- (1) the base institution being hybridised, using the name of a known logic in HETS or even of one of its sublogics, written in HETS syntax as `LogicName.SublogicName`,
- (2) the kind of symbols allowed to appear in a quantification, which can be `nominal` or a kind of symbols of the base institution, referred to by its HETS name,
- (3) the constraints made on the models of the hybridised institution, which can be of two kinds: on the accessibility relations between possible worlds (reflexive, transitive etc.) or on the interpretation of symbols of a certain kind (universes, nominals, or a kind from the base institution) in the possible worlds.

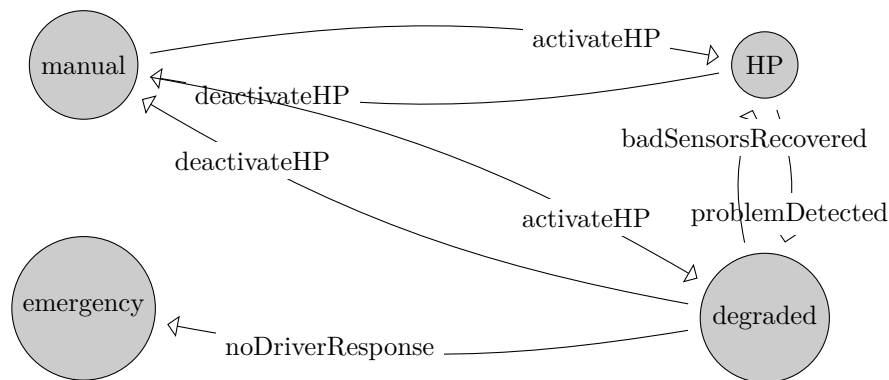
Listing 1 shows how the hybridisation of the extension of the CASL logic with rigid symbols, that will be used in the example in Sec. 4, is specified¹: first we define `HRigidCASL` as the hybridisation of `RigidCASL` with quantification on rigid constants and nominals. After this definition is analyzed by HETS, it is recorded for further extensions: `HRigidCASLC` adds the constraints that rigid sorts, rigid predicates and rigid total functions share the interpretation and rigid partial functions share the domain of definition in each possible world of a model. HETS generates for each of these two definitions a new instance of the class `Logic`, which will become available for specification in HETS once the newly generated code is compiled. The resulting logic will inherit the syntax of the base institution for declarations of base symbols and for base sentences, and will use generic syntax for declarations of nominals and modalities, and for hybrid sentences. Full details of the syntax are available at [2].

■ **Listing 1** Hybridisation of `RigidCASL`.

```
newhlogic HRigidCASL =
  base: RigidCASL .
  quant: rigid const, nominal .
end
newhlogic HRigidCASLC =
  hlogic: HRigidCASL .
  constr: SameInterpretation(rigid sort),      SameInterpretation(rigid op),
          SameInterpretation(rigid pred),      SameDomain(rigid partial) .
end
```

The process of lifting a comorphism to first-order logic from a base institution to its hybridisation has only two parameters: the HETS name of the comorphism being lifted and the name of the hybridisation of the base institution that will be the source of the lifted comorphism. The latter is needed because a base institution admits more than one hybridisation. Again, HETS analyses this definition and generates source code that must be compiled to make the comorphism available for translation and proofs by translation.

¹ More examples can be found at <https://ontohub.org/forver>.



■ **Figure 1** The modes and reconfigurations of the highway pilot.

■ **Listing 2** Lifting the translation to first-order logic of RigidCASL to HRigidCASLC.

```

newhcomorphism HRigid2CASL =
basecomorphism: Rigid2CASL
sourcehlogic: HRigidCASLC
end

```

4 Case Study: specification and verification of a highway pilot

We now discuss the example of an autonomous driving system for passenger cars and heavy trucks, called highway pilot (HP). The problem description is adapted from [16]. HP can only be activated when driving on a highway. After activation, the electronic system will keep driving the car on the highway, relying on information from radar and camera sensors. These sensors may exhibit faults or may fail to give a correct interpretation of the surroundings, depending on weather, traffic and road conditions. Unlike bad conditions, faults will not disappear after some time unless the sensors are physically repaired. Faults and bad conditions may be undetected for some time. If a fault or bad conditions are detected when HP is on, the system will enter a so-called **degraded** driving mode, where a safer driving style is adopted, typically including driving slower, and the driver is alerted that he/she should take over driving. If the driver does not deactivate HP mode within a time limit after being alerted, an emergency stop will be performed. If the bad conditions disappear (all unreliable sensors become reliable) before the time limit for HP deactivation is reached and there are no faulty sensors, the system will return to HP mode and stop alerting the driver.

The modes of the systems and the events causing changes of modes are depicted in Fig. 1. In Listing 3 we show how they are specified, together with axioms stating that there are no other modes, that the system can change from the manual to HP and degraded modes and that the only transitions from the modes HP and degraded along the reconfiguration deactivateHP are to the manual mode.

■ **Listing 3** Modes.

```

nominals manual, hp, degraded, emergency
modalities activateHP, deactivateHP, problemDetected,
           noDriverResponse, badSensorsRecovered : 2

. manual \/ hp \/ degraded \/ emergency    %(no_other_states)%
. @ manual : <activateHP> (hp \/ degraded) %(manual_to_hp_or_degraded)%
. not (manual \/ emergency)
  => <deactivateHP> manual /\ [deactivateHP] manual %(back_to_manual)%

```

17:6 Hybridisation of Institutions in HETS

We also keep track of the current state of the HP system, using a different transition system than the one between modes. The system states are loosely specified with the help of observers for the current speed, the current status of the sensors (working, detected bad conditions, detected fault, undetected bad conditions, undetected faults), flags for checking whether the driver alert is on or off and if the driver has turned the HP off and a time counter for checking the time limit in the degraded mode. The transitions between states are given by a non-rigid predicate `step`. In each mode, transitions are possible only from the states that are valid in that mode, as defined by a non-rigid predicate `isValid`. Recall that non-rigid symbols admit different interpretations in different possible worlds. Listing 4 shows the definition of valid states for the mode `hp`: those reached after HP was activated from `manual` mode, those reached after bad sensors recovered in `degraded` mode and those that are reached from a valid state in the `hp` mode and do not satisfy the conditions for reconfigurations.²

■ **Listing 4** Highway pilot mode.

```
. @ hp : forallH S' : State . isValid(S') <=> crtDeactivateTime(S') = 0 /\
  (( existsH S : State . @ manual :
    isValid(S) /\ step(S, S') /\ activateHPWorkingCond(S'))
  \/ ( existsH S : State . @ degraded :
    isValid(S) /\ step(S, S') /\ badSensorsRecoveredCond(S'))
  \/ ( existsH S : State . @ hp :
    isValid(S) /\ step(S, S') /\
    not hpDeactivated(S') /\ not problemDetectedCond(S')))
%(def_isValid_hp)%
```

The other modes are specified in a similar way³. We can now verify that the valid states of each mode have the expected properties. The corresponding sentence for the mode `hp`, stating that all valid states have no detected faulty or unreliable sensors is shown in Listing 5. We can prove this conjecture in HETS by translation, using the SPASS prover with a time limit of 70 seconds on a modern machine. Proving that in every state valid in `degraded` mode there is at least one faulty or unreliable sensor takes significantly more time (time limit of 500 seconds with SPASS) and requires the introduction of a lemma. This is typical for proofs in first-order logic, especially in the case of very large theories as the one obtained in this case via translation.

■ **Listing 5** Conjectures.

```
. forallH S : State . (@ hp : isValid(S))
=> not exists s : Sensor .
  status(S, s) = detectedBad \/ status(S,s) = detectedFault
%(no_detected_problems_hp)% %implied
```

5 Conclusions and future work

By implementing in HETS the hybridisation method and the lifting of translations introduced in [5, 6], we obtain a framework for specification and verification of reconfigurable systems. Proofs are done by translation to first-order logic using the first-order provers already integrated with HETS. Given the large variety of hybrid institutions that can be specified, this is often the only tool support available. An interesting enterprise would be to implement

² Note that `forallH` and `existsH` are the universal and existential quantifiers introduced via hybridisation; their semantics does not always subsume that of quantification in the base logic, see [5].

³ The complete specification of the HP system is available under <https://ontohub.org/forver/hp.dol>.

a generic parameterized prover for hybrid logics, possibly following the ideas of [1], and to make it available in HETS for each generated hybridised institution. The results of [15] hold for a restricted form of hybridisation, without quantifications on nominals and with no constraints on models and therefore cannot be applied in our setting.

References

- 1 D. Găină. Birkhoff style calculi for hybrid logics. *Formal Asp. Comput.*, 29(5):805–832, 2017. doi:10.1007/s00165-016-0414-y.
- 2 M. Codescu and R. Diaconescu. Hspec language definition. URL: <http://imar.ro/~diacon/forver/Hdef.pdf>.
- 3 M. Codescu and R. Diaconescu. The H system. <http://imar.ro/~diacon/forver/forver.html>.
- 4 R. Diaconescu. Grothendieck Institutions. *Applied Categorical Structures*, 10(4):383–402, 2002. doi:10.1023/A:1016330812768.
- 5 R. Diaconescu. Quasi-varieties and initial semantics for hybridized institutions. *J. Log. Comput.*, 26(3):855–891, 2016. doi:10.1093/logcom/ext016.
- 6 R. Diaconescu and A. Madeira. Encoding hybridized institutions into first-order logic. *Mathematical Structures in Computer Science*, 26(5):745–788, 2016. doi:10.1017/S0960129514000383.
- 7 R. Diaconescu and P. S. Stefaneas. Ultraproducts and possible worlds semantics in institutions. *Theor. Comput. Sci.*, 379(1-2):210–230, 2007. doi:10.1016/j.tcs.2007.02.068.
- 8 J. A. Goguen and R. M. Burstall. Institutions: Abstract Model Theory for Specification and Programming. *Journal of the Association for Computing Machinery*, 39:95–146, 1992. doi:10.1145/147508.147524.
- 9 A. Madeira, R. Neves, L. Soares Barbosa, and M. A. Martins. A method for rigorous design of reconfigurable systems. *Sci. Comput. Program.*, 132:50–76, 2016. doi:10.1016/j.scico.2016.05.001.
- 10 M. A. Martins, A. Madeira, R. Diaconescu, and L. Soares Barbosa. Hybridization of Institutions. In *CALCO*, volume 6859 of *Lecture Notes in Computer Science*, pages 283–297. Springer, 2011. doi:10.1007/978-3-642-22944-2_20.
- 11 J. Meseguer. General logics. In H.-D. Ebbinghaus et al., editors, *Proceedings, Logic Colloquium, 1987*, pages 275–329. North-Holland, 1989.
- 12 T. Mossakowski. Comorphism-based Grothendieck logics. In K. Diks and W. Rytter, editors, *Mathematical foundations of computer science*, volume 2420 of *Lecture Notes in Computer Science*, pages 593–604. Springer Verlag, London, 2002. doi:10.1007/3-540-45687-2_49.
- 13 T. Mossakowski, C. Maeder, and K. Lüttich. The Heterogeneous Tool Set. In O. Grumberg and M. Huth, editors, *TACAS 2007*, volume 4424 of *Lecture Notes in Computer Science*, pages 519–522. Springer, Heidelberg, 2007. doi:10.1007/978-3-540-71209-1_40.
- 14 R. Neves, A. Madeira, M. A. Martins, and L. Soares Barbosa. Hybridisation at Work. In R. Heckel and S. Milius, editors, *CALCO 2013*, volume 8089 of *Lecture Notes in Computer Science*, pages 340–345. Springer, 2013. doi:10.1007/978-3-642-40206-7_28.
- 15 R. Neves, A. Madeira, M. A. Martins, and L. Soares Barbosa. Proof theory for hybrid(ised) logics. *Sci. Comput. Program.*, 126:73–93, 2016. doi:10.1016/j.scico.2016.03.001.
- 16 M. Nyberg. Safety analysis of autonomous driving using semi-Markov processes. In Stein Haugen, Anne Barros, Coen van Gulijk, Trond Kongsvik, and Jan Erik Vinnem, editors, *Safety and Reliability—Safe Societies in a Changing World*, pages 781–788. CRC Press, 2018. doi:10.1201/9781351174664-97.

A Implementation

In this appendix we give an overview of how the hybridisation method was implemented in HETS. We have made some simplifications and changes of the actual names used in the HETS source code to ease understanding.

A.1 Adding a new logic in HETS

HETS has an abstract interface for logics, in the form of a Haskell multiparameter type class with functional dependencies, called `Logic`. The parameters are types for the constituents of a logic: its identifier, its signatures and signature morphisms, its symbols of signatures with their kinds, its low-level, human readable syntax in the form of basic specifications for theories, lists of symbols for convenient use during structuring and symbol maps for signature morphisms, its sublogics and its proof trees. Being a type class, `Logic` provides a list of methods that must be provided for each particular choice of the parameter types in order to obtain a new instance of the type class. These include, among others, composition of signature morphisms, parsers, printers, static analysis of basic specifications, symbol lists and symbol maps, sentence translation along signature morphisms, various operations on signatures and signature morphisms. The functional dependency between the type logic identifier and all other types is used to determine the missing types and thus the correct instance of a function in the type class. This means that all methods in the class `Logic` take as first argument the logic identifier, and this determines the logic uniquely.

To sketch an example, propositional logic has as identifier a singleton type, called `Propositional`. Types must be provided for its constituents: signatures are sets of names (implemented in HETS using the datatype `Id`) of propositional symbols, signature morphisms are maps between these sets, where we also store the source and the target signature for each morphism, and so on.

■ **Listing 6** Signatures in propositional logic.

```
newtype PropSign = PropSign {items :: Set Id}
```

The type class `Logic` contains a method for union of signatures: `signature_union :: lid -> sign -> sign -> Result sign`, where the type `Result a` is a polymorphic type with variable `a` used for dealing with errors (the union of signature may not give a legal signature for each institution). We must provide an implementation of this method for propositional logic, and this will be done as a method `signatureUnion :: PropSign -> PropSign -> Result PropSign`. Then we must provide an instance declaration for the types for components of propositional logic where we say how the methods of the type class are implemented.

■ **Listing 7** Propositional logic.

```
instance Logic -- the type class
  Propositional -- the logic identifier
  ...
  PropSign -- the type of signatures
  PropMorphism
  ...
where
  ...
  signature_union Propositional = signatureUnion
  ...
```


To sum up, adding a new logic in HETS requires creating a new instance of the `Logic` class, and this is achieved by defining types for the constituents of that logic and by implementing the methods of the `Logic` class for these types.

A.2 Generic implementation of hybridisation in HETS

The first step is to define the generic types for the constituents of a hybridised institution. We used type variables for the parts that come from the base institution. For example, the type of hybrid signatures is presented in Listing 8.

■ **Listing 8** Hybrid signatures.

```
data HSign sig = HSign {
    baseSig :: sig,
    noms    :: Set Id,
    mods    :: Set Id}
```

where the variable `sig` stands for the base signatures. Then we need to implement the methods of the `Logic` class over these generic types. Typically this will require that the corresponding method in the base institution is involved, and we need to make it accessible. This is achieved by giving as an argument the logic identifier of the base institution and imposing the condition that the type variables that appear in the generic types introduced at the first step will be instantiated with the corresponding types from the base institution. For example the method for union of hybrid signatures presented in Listing 9 will have to make the union of the base signatures. We add the requirement that the argument for the type variable `sig` is the type of signatures of the base institution, as recorded in the Haskell context of the method `sigUnion`. The identifier `baseLid` allows us to properly identify the `Logic` instance of the base institution, and thus `signature_union baseLid` will invoke the implementation of signature union in the base institution, for the base signatures of the hybrid signatures that we want to unite. Then we unite the sets of nominals and modalities, respectively, and return the result.

■ **Listing 9** Union of hybrid signatures.

```
sigUnion :: (Logic baseLid ... sig ...)
          => baseLid -> HSign sig -> HSign sig -> Result (HSign sig)
sigUnion baseLid hsig1 hsig2 = do
  usig <- signature_union baseLid (baseSig hsig1) (baseSig hsig2)
  let uNoms = Set.union (noms hsig1) (noms hsig2)
      uMods = Set.union (mods hsig1) (mods hsig2)
  return $ HSign usig uNoms uMods
```

As a result of these two steps, we obtain generic types for hybrid institutions and generic implementations of the methods in the `Logic` class for these types. Let us assume we want to extend HETS with the hybridisation of propositional logic, with no quantification and no semantic constraints on models. This is written as in Listing 10.

■ **Listing 10** Hybridisation of propositional logic.

```
newhlogic HProp =
  base: Propositional .
end
```

When HETS analyzes this definition, it generates a new instance of the `Logic` class, whose component type for signatures is `HSign PropSign`, and similarly for the other component types of a logic. The instance declaration in Listing 11 states that the implementation of signature union for the new logic `HProp` is given by the method `sigUnion` introduced in

17:10 Hybridisation of Institutions in HETS

Listing 9. It uses partial application: the methods on both sides of the equal sign take as arguments two hybrid signatures. `HProp` is the unique value of the singleton type `HProp` generated as a logic identifier for the new hybridised logic that we want to define.

■ **Listing 11** Logic instance for hybrid propositional logic.

```
instance Logic
  HProp
  ...
  (HSign PropSign)
  ...
where
  ...
  signature_union HProp = sigUnion Propositional
  ...
```

Nominal String Diagrams

Samuel Balco

Department of Informatics, University of Leicester, United Kingdom

<https://gdlyrtnap.pl>

sb782@leicester.ac.uk

Alexander Kurz

Department of Computer Science, Chapman University, Orange California, USA

akurz@chapman.edu

Abstract

We introduce nominal string diagrams as string diagrams internal in the category of nominal sets. This requires us to take nominal sets as a monoidal category, not with the cartesian product, but with the separated product. To this end, we develop the beginnings of a theory of monoidal categories internal in a symmetric monoidal category. As an instance, we obtain a notion of a nominal PROP as a PROP internal in nominal sets. A 2-dimensional calculus of simultaneous substitutions is an application.

2012 ACM Subject Classification Software and its engineering → General programming languages; Theory of computation → Models of computation; Theory of computation → Logic; Mathematics of computing

Keywords and phrases string diagrams, nominal sets, separated product, simultaneous substitutions, internal category, monoidal category, internal monoidal categories, PROP

Digital Object Identifier 10.4230/LIPIcs.CALCO.2019.18

Acknowledgements We are grateful to Fredrik Dahlqvist, Giuseppe Greco, Samuel Mimram, Drew Moshier, Alessandra Palmigiano, David Pym, Mike Shulman, Pawel Sobocinski, Thomas Streicher, Georg Struth, Apostolos Tzimoulis and Fabio Zanasi for discussions on the topic of this paper.

1 Introduction

One reason for the success of string diagrams, see [19] for an overview, can be formulated by the slogan “only connectivity matters” [4, Sec.10.1]. Technically, this is usually achieved by ordering input and output wires and using their ordinal numbers as implicit names. We write $\underline{n} = \{1, \dots, n\}$ to denote the set of n numbered wires and $f : \underline{n} \rightarrow \underline{m}$ for diagrams f with n inputs and m outputs. The approach of using order to implicitly name wires is particularly convenient for the generalisations of Lawvere theories known as PROPs [16]. In particular, the paper on composing PROPs [13] has been influential [2, 3].

On the other hand, if only connectivity matters, it is natural to consider a formalisation of PROPs in which wires are not ordered. Thus, instead of ordering wires, we fix a countably infinite set \mathcal{N} of “names” a, b, \dots , on which the only supported operation or relation is equality. Mathematically, this means that we work internally in the category of nominal sets introduced by Gabbay and Pitts [8, 18]. In the remainder of the introduction, we highlight some of the features of this approach.

Partial commutative vs total symmetric tensor. One reason why ordered names are convenient is that the tensor \oplus is given by the categorical coproduct (addition) in the skeleton \mathbb{F} of the category of finite sets. Even though $\underline{n} \oplus \underline{m} = \underline{m} \oplus \underline{n}$ on objects, the tensor is not commutative but only symmetric, since the canonical arrow $\underline{n} \oplus \underline{m} \rightarrow \underline{m} \oplus \underline{n}$ is not the identity.



© Samuel Balco and Alexander Kurz;

licensed under Creative Commons License CC-BY

8th Conference on Algebra and Coalgebra in Computer Science (CALCO 2019).

Editors: Markus Roggenbach and Ana Sokolova; Article No. 18; pp. 18:1–18:20

Leibniz International Proceedings in Informatics



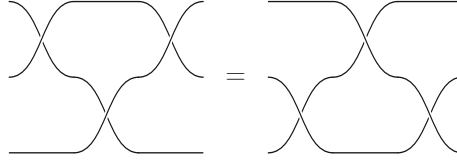
LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

18:2 Nominal String Diagrams

On the other hand, in the category \mathfrak{nF} of finite subsets of \mathcal{N} (which is equivalent to \mathfrak{F} as an ordinary category), there is a commutative tensor $A \uplus B$ given by union of disjoint sets. The feature that makes commutativity possible is that \uplus is partial with $A \uplus B$ defined if and only if $A \cap B = \emptyset$.

While it would be interesting to develop a general theory of partially monoidal categories, our approach in this paper is based on the observation that the partial operation $\uplus : \mathfrak{nF} \times \mathfrak{nF} \rightarrow \mathfrak{nF}$ is a total operation $\uplus : \mathfrak{nF} * \mathfrak{nF} \rightarrow \mathfrak{nF}$ where $*$ is the separated product of nominal sets [18].

Symmetries disappear in 3 dimensions. From a graphical point of view, the move from ordered wires to named wires corresponds to moving from planar graphs to graphs in 3 dimensions. Instead of having a one dimensional line of inputs or outputs, wires are now sticking out of a plane [12]. As a benefit there are no wire-crossings, or, more technically, there are no symmetries to take care of. This simplifies the rewrite rules of calculi formulated in the named setting. For example, rules such as



are not needed anymore. For more on this compare Figs 3 and 4.

Example: Simultaneous Substitutions. Substitutions $[a \mapsto b]$ can be composed sequentially and in parallel as in

$$[a \mapsto b]; [b \mapsto c] = [a \mapsto c] \quad [a \mapsto b] \uplus [c \mapsto d] = [a \mapsto b, c \mapsto d].$$

We call \uplus the tensor, or the monoidal or vertical or parallel composition. Semantically, the simultaneous substitution on the right-hand side above, will correspond to the function $f : \{a, c\} \rightarrow \{b, d\}$ satisfying $f(a) = b$ and $f(c) = d$. Importantly, parallel composition of simultaneous substitutions is partial. For example, $[a \mapsto b] \uplus [a \mapsto c]$ is undefined, since there is no function $\{a\} \rightarrow \{b, c\}$ that maps a simultaneously to both b and c .

The advantages of a 2-dimensional calculus for simultaneous substitutions over a 1-dimensional calculus are the following. A calculus of substitutions is an algebraic representation, up to isomorphism, of the category \mathfrak{nF} of finite subsets of \mathcal{N} . In a 1-dimensional calculus, operations $[a \mapsto b]$ have to be indexed by finite sets S

$$[a \mapsto b]_S : S \cup \{a\} \rightarrow S \cup \{b\}$$

for sets S with $a, b \notin S$. On the other hand, in a 2-dimensional calculus with an explicit operation \uplus for set union, indexing with subsets S is unnecessary. Moreover, while the swapping

$$\{a, b\} \rightarrow \{a, b\}$$

in the 1-dimensional calculus needs an auxiliary name such as c in $[a \rightarrow c]_{\{b\}}; [b \rightarrow a]_{\{c\}}; [c \rightarrow a]_{\{b\}}$ it is represented in the 2-dimensional calculus directly by

$$[a \rightarrow b] \uplus [b \rightarrow a]$$

Finally, while it is possible to write down the equations and rewrite rules for the 1-dimensional calculus, it does not appear as particularly natural. In particular, only in the 2-dimensional calculus, will the swapping have a simple normal form such as $[a \rightarrow b] \uplus [b \rightarrow a]$ (unique up to commutativity of \uplus).

Overview. In order to account for partial tensors, Section 3 develops the notion of a monoidal category internal in a monoidal category. Section 4 is devoted to examples, while Section 5 introduces the notion of a nominal PROP and Section 6 shows that the categories of ordinary and of nominal PROPs are equivalent.

2 Setting the Scene: String Diagrams and Nominal Sets

We review some of the terminology but need to refer to the literature for details.

2.1 String Diagrams and PROPs

String diagrams are a 2-(or higher)-dimensional notation for monoidal categories [12]. Their algebraic theory can be formalised by PROPs as defined by MacLane [15]. There is also the weaker notion by Lack [13], see Remark 2.9 of Zanasi [22] for a discussion.

A PROP (*products and permutation category*) is a symmetric strict monoidal category, with natural numbers as objects, where the monoidal tensor \oplus is addition. Moreover, PROPs, along with strict symmetric monoidal functors, that are identities on objects, form the category PROP. A PROP contains all bijections between numbers as they can be generated from the symmetry (twist) $\sigma : 1 \oplus 1 \rightarrow 1 \oplus 1$ and from the parallel composition \oplus and sequential composition $;$ (which we write in diagrammatic order). We denote by $\sigma_{n,m}$ the canonical symmetry $n \oplus m \rightarrow m \oplus n$. Functors between PROPs preserve bijections.

PROPs can be presented in algebraic form by operations and equations as *symmetric monoidal theories* (SMTs) [22].

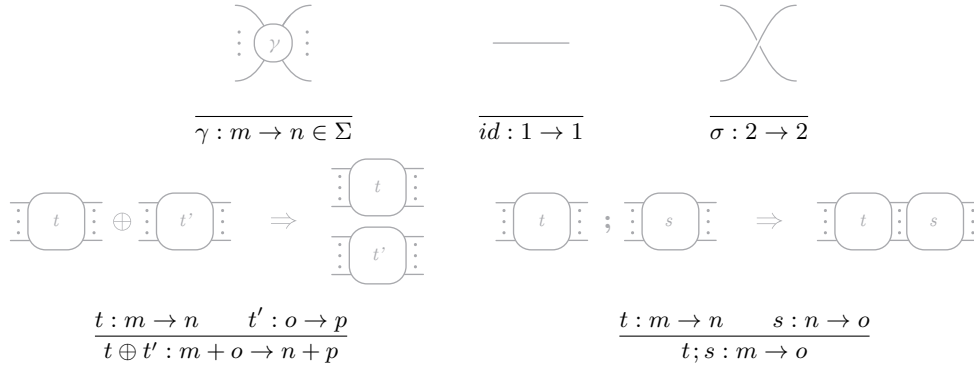
An SMT (Σ, E) has a set Σ of generators, where each generator $\gamma \in \Sigma$ is given an arity m and co-arity n , usually written as $\gamma : m \rightarrow n$ and a set E of equations, which are pairs of Σ -terms. Σ -terms can be obtained by composing generators in Σ with the unit $id : 1 \rightarrow 1$ and symmetry $\sigma : 2 \rightarrow 2$, using either the parallel or sequential composition (see Fig 1). Equations E are pairs of Σ -terms with the same arity and co-arity.

Given an SMT (Σ, E) , we can freely generate a PROP, by taking Σ -terms as arrows, modulo

- the equations stating that, together with id , the compositions $;$ and \oplus form monoids
- the equations of Fig 2
- the equations E

PROPs have a nice 2-dimensional notation, where sequential composition is horizontal composition of diagrams, and parallel/tensor composition is vertical stacking of diagrams (see Fig 1). We now present the SMTs of bijections \mathbb{B} , injections \mathbb{I} , surjections \mathbb{S} ,

18:4 Nominal String Diagrams



■ **Figure 1** SMT Terms.

$$\begin{aligned} \sigma_{1,1}; \sigma_{1,1} &= id_2 && \text{(SMT-sym)} \\ (s;t) \oplus (u;v) &= (s \oplus u); (t \oplus v) && \text{(SMT-ch)} \\ (t \oplus id_z); \sigma_{n,z} &= \sigma_{m,z}; (id_z \oplus t) && \text{(SMT-nat)} \end{aligned}$$

■ **Figure 2** Equations of symmetric monoidal categories.

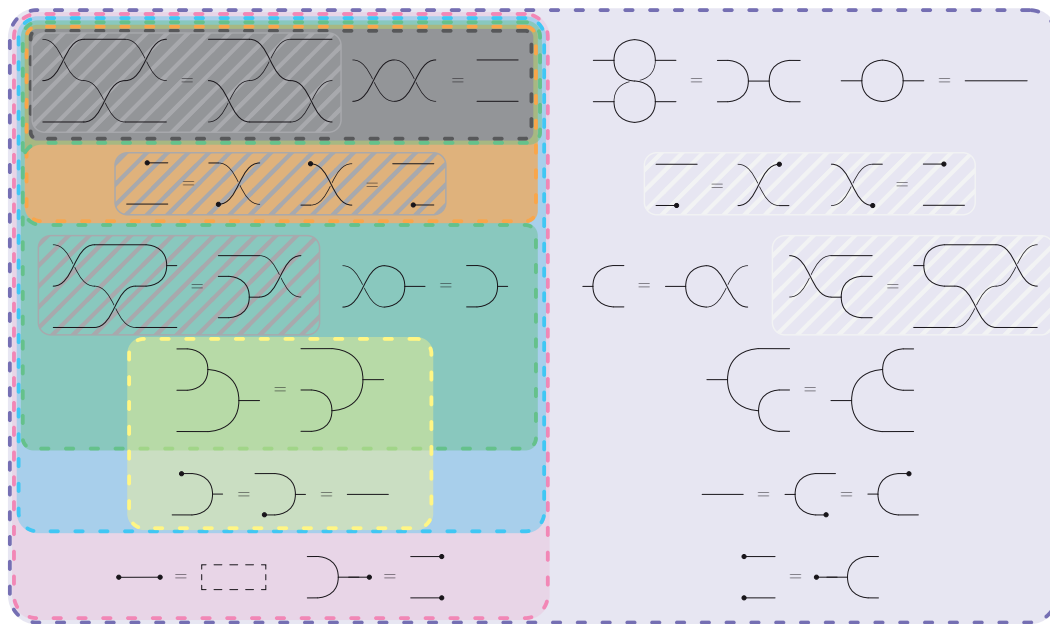
functions \mathbb{F} , partial functions \mathbb{P} , relations \mathbb{R} and monotone maps \mathbb{M} .¹ The diagram in Fig 3 shows the generators and the equations that need to be added to the empty SMT, to get a presentation of the given theory. To ease comparison with the corresponding nominal monoidal theories in Fig 4 later we also added on a striped background the equations for wire-crossings that are already implied by the naturality of symmetries (SMT-nat). These are equations that are part of the definition of a PROP in the sense of MacLane [15] but not in the sense of Lack [13]. The right-hand equation for bijections \mathbb{B} is (SMT-sym) and holds in all symmetric monoidal theories. We list it here to emphasise the difference with Fig 4.

2.2 Nominal Sets

Let \mathcal{N} be a countably infinite set of “names” or “atoms”. Let \mathfrak{S} be the group of finite² permutations $\mathcal{N} \rightarrow \mathcal{N}$. An element $x \in X$ of a group action $\mathfrak{S} \times X \rightarrow X$ is supported by $S \subseteq \mathcal{N}$ if $\pi \cdot x = x$ for all $\pi \in \mathfrak{S}$ such that π restricted to S is the identity. A group action $\mathfrak{S} \times X \rightarrow X$ such that all elements of X have finite support is called a *nominal set*. We write $\text{supp}(x)$ for the minimal support of x and say that a is fresh for x if $a \notin \text{supp}(x)$. Nom for the category of nominal sets, which has as maps the *equivariant* functions, that is, those functions that respect the permutation action. Our main example is the category of simultaneous substitutions:

¹ The theory of monotone maps \mathbb{M} does not include equations involving the symmetry σ and is in fact presented by a so-called PRO rather than a PROP. However, in this paper we will only be dealing with theories presented by PROPs (the reason why this is the case is illustrated in the proof of Proposition 20).

² A permutation is called finite if it is generated by finitely many transpositions.



■ **Figure 3** Symmetric monoidal theories (compiled from [14]).

► **Example 1** (\mathbf{nF}). We denote by \mathbf{nF} the category of finite subsets of \mathcal{N} with all functions. While \mathbf{nF} is a category, it also carries additional nominal structure. In particular, both the set of objects and the set of arrows are nominal sets with $\text{supp}(A) = A$ and $\text{supp}(f) = A \cup B$ for $f : A \rightarrow B$. The categories of injections, surjections, bijections, partial functions and relations are further examples along the same lines.

3 Internal monoidal categories

We introduce the notion of an internal monoidal category. Given a symmetric monoidal category $(\mathcal{V}, I, \otimes)$ with finite limits, we are interested in categories \mathbb{C} , internal in \mathcal{V} , that carry a monoidal structure not of type $\mathbb{C} \times \mathbb{C} \rightarrow \mathbb{C}$ but of type $\mathbb{C} \otimes \mathbb{C} \rightarrow \mathbb{C}$. This will allow us to account for the partiality of \uplus discussed in the introduction. We present our motivating example before we give Definition 11.

► **Example 2.**

- The symmetric monoidal (closed) category $(\mathbf{Nom}, 1, *)$ of nominal sets with the separated product $*$ is defined as follows [18]. 1 is the terminal object, i.e. a singleton with empty support. The separated product of two nominal sets is defined as $A * B = \{(a, b) \in A \times B \mid \text{supp}(a) \cap \text{supp}(b) = \emptyset\}$.
- The category \mathbf{nF} of Example 1 is an internal monoidal category with monoidal operation given by $A \uplus B = A \cup B$ if A, B are disjoint and $f \uplus f' = f \cup f'$ if A, A' and B, B' are disjoint where $f : A \rightarrow B$ and $f' : A' \rightarrow B'$.

$(\mathbf{nF}, \emptyset, \uplus)$ as defined in the previous example is not a monoidal category, since \uplus , being partial, is not an operation of type $\mathbf{nF} \times \mathbf{nF} \rightarrow \mathbf{nF}$. The purpose of this section is to define the notion of internal monoidal category and to show that $(\mathbf{nF}, \emptyset, \uplus)$ is an internal monoidal category in $(\mathbf{Nom}, 1, *)$ with \uplus of type

18:6 Nominal String Diagrams

$$\uplus : \mathfrak{nF} * \mathfrak{nF} \rightarrow \mathfrak{nF}.$$

To this end we need to extend $*$: $\mathbf{Nom} \times \mathbf{Nom} \rightarrow \mathbf{Nom}$ to

$$* : \mathbf{Cat}(\mathbf{Nom}) \times \mathbf{Cat}(\mathbf{Nom}) \rightarrow \mathbf{Cat}(\mathbf{Nom})$$

where we denote by $\mathbf{Cat}(\mathbf{Nom})$, the category of (small) internal categories in \mathbf{Nom} . The necessary (and standard) notation from internal categories is reviewed in Appendix A.

► **Remark 3.** Let \mathbb{C} be an internal category in a symmetric monoidal category $(\mathcal{V}, I, \otimes)$ with finite limits. Since \otimes need not preserve finite limits, we cannot expect that defining $(\mathbb{C} \otimes \mathbb{C})_0 = \mathbb{C}_0 \otimes \mathbb{C}_0$ and $(\mathbb{C} \otimes \mathbb{C})_1 = \mathbb{C}_1 \otimes \mathbb{C}_1$ results in $\mathbb{C} \otimes \mathbb{C}$ being an internal category.

Consequently, putting $(\mathbb{C} \otimes \mathbb{C})_1 = \mathbb{C}_1 \otimes \mathbb{C}_1$ does not extend \otimes to an operation $\mathbf{Cat}(\mathcal{V}) \times \mathbf{Cat}(\mathcal{V}) \rightarrow \mathbf{Cat}(\mathcal{V})$. To show what goes wrong in a concrete instance is the purpose of the next example.

► **Example 4.** Define a binary operation $\mathfrak{nF} * \mathfrak{nF}$ as $(\mathfrak{nF} * \mathfrak{nF})_0 = \mathfrak{nF}_0 * \mathfrak{nF}_0$ and $(\mathfrak{nF} * \mathfrak{nF})_1 = \mathfrak{nF}_1 * \mathfrak{nF}_1$. Then $\mathfrak{nF} * \mathfrak{nF}$ cannot be equipped with the structure of an internal category. Indeed, assume for a contradiction that there was an appropriate pullback $(\mathfrak{nF} * \mathfrak{nF})_2$ and arrow $comp$ such that the two diagrams commute:

$$\begin{array}{ccc} (\mathfrak{nF} * \mathfrak{nF})_2 & \xrightarrow{\quad comp \quad} & \mathfrak{nF}_1 * \mathfrak{nF}_1 \\ \pi_1 \downarrow \pi_2 & & \downarrow \begin{array}{l} dom \\ cod \end{array} \\ \mathfrak{nF}_1 * \mathfrak{nF}_1 & \xrightarrow[\quad cod \quad]{\quad dom \quad} & \mathfrak{nF}_0 * \mathfrak{nF}_0 \end{array}$$

Let $\delta_{xy} : \{x\} \rightarrow \{y\}$ be the unique function in \mathfrak{nF} of type $\{x\} \rightarrow \{y\}$. Then $((\delta_{ac}, \delta_{bd}), (\delta_{cb}, \delta_{da}))$, which can be depicted as

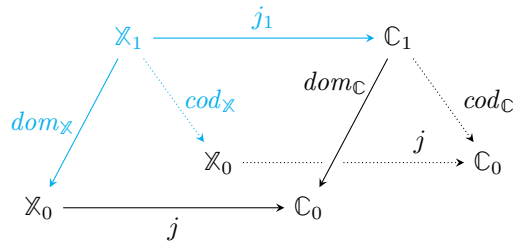
$$\begin{array}{ccc} \{a\} & \xrightarrow{\delta_{ac}} & \{c\} & \xrightarrow{\delta_{cb}} & \{b\} \\ \{b\} & \xrightarrow{\delta_{bd}} & \{d\} & \xrightarrow{\delta_{da}} & \{a\} \end{array}$$

is in the pullback $(\mathfrak{nF} * \mathfrak{nF})_2$, but there is no $comp$ such that the two squares above commute, since $comp((\delta_{ac}, \delta_{bd}), (\delta_{cb}, \delta_{da}))$ would have to be $(\delta_{ab}, \delta_{ba})$, which do not have disjoint support and therefore are not in $\mathfrak{nF}_1 * \mathfrak{nF}_1$. ◀

The solution to the problem consists in assuming that the given symmetric monoidal category with finite limits $(\mathcal{V}, 1, \otimes)$ is semi-cartesian (aka affine), that is, the unit 1 is the terminal object. In such a category there are canonical arrows natural in A and B

$$j : A \otimes B \rightarrow A \times B$$

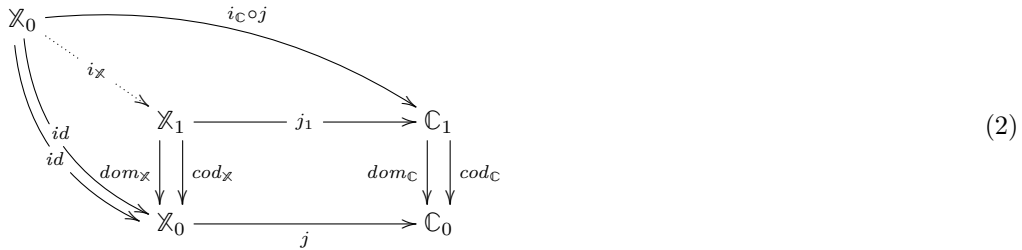
and we can use them to define arrows $j_1 : (\mathbb{C} \otimes \mathbb{C})_1 \rightarrow \mathbb{C}_1 \times \mathbb{C}_1$ that give us the right notion of tensor on arrows. From our example \mathfrak{nF} above, we know that we want arrows (f, g) to be in $(\mathbb{C} \otimes \mathbb{C})_1$ if $dom(f) \cap dom(g) = \emptyset$ and $cod(f) \cap cod(g) = \emptyset$. We now turn this into a category theoretic definition, which, in fact, is an instance of the general and well-known construction of pulling back an internal category \mathbb{C} along an arrow $j : X \rightarrow \mathbb{C}_0$ to yield an internal category \mathbb{X} with $\mathbb{X}_0 = X$ and \mathbb{X}_1 the pullback of $\langle dom_{\mathbb{C}}, cod_{\mathbb{C}} \rangle$ along $j \times j$, or, equivalently, the limit in the following diagram



which we abbreviate to

$$\begin{array}{ccc}
 X_1 & \xrightarrow{j_1} & C_1 \\
 \text{dom}_X \downarrow & & \downarrow \text{dom}_C \\
 X_0 & \xrightarrow{j} & C_0
 \end{array}
 \quad \begin{array}{ccc}
 & \text{cod}_X \downarrow & \\
 & & \downarrow \text{cod}_C \\
 & &
 \end{array}
 \quad (1)$$

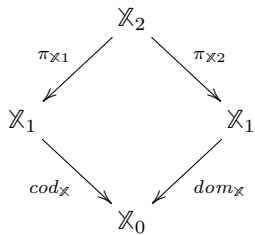
Next we define $i : X_0 \rightarrow X_1$ as the arrow into the limit X_1 given by



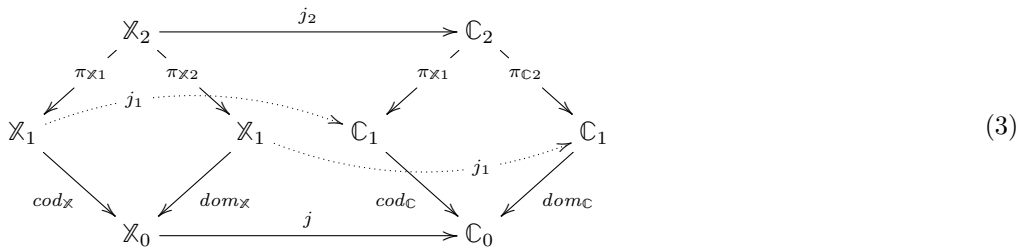
from which one reads off

$$\text{dom}_X \circ i_X = \text{id}_{X_0} = \text{cod}_X \circ i_X$$

Next, X_2 is the pullback



Recalling the definition of j_1 from (1), there is also a corresponding $j_2 : X_2 \rightarrow C_2$ due to the fact that the product of pullbacks is a pullback of products.



18:8 Nominal String Diagrams

Recall the definition of the limit \mathbb{X}_1 from (1). Then $comp_{\mathbb{X}} : \mathbb{X}_2 \rightarrow \mathbb{X}_1$ is the arrow into \mathbb{X}_1

$$\begin{array}{ccccc}
 \mathbb{X}_2 & & & & \\
 \downarrow \scriptstyle{dom_{\mathbb{X}} \circ \pi_{\mathbb{X}_1}} & \searrow \scriptstyle{comp_{\mathbb{X}}} & & \searrow \scriptstyle{comp_{\mathbb{C}} \circ j_2} & \\
 \mathbb{X}_1 & \xrightarrow{\quad j_1 \quad} & \mathbb{C}_1 & & \\
 \downarrow \scriptstyle{dom_{\mathbb{X}}} & & \downarrow \scriptstyle{dom_{\mathbb{C}}} & & \\
 \mathbb{X}_0 & \xrightarrow{\quad j \quad} & \mathbb{C}_0 & & \\
 & & \downarrow \scriptstyle{cod_{\mathbb{C}}} & & \\
 & & \mathbb{C}_0 & &
 \end{array}
 \quad (4)$$

from which one reads off

$$dom_{\mathbb{X}} \circ comp_{\mathbb{X}} = dom_{\mathbb{X}} \circ \pi_{\mathbb{X}_1} \quad cod_{\mathbb{X}} \circ comp_{\mathbb{X}} = cod_{\mathbb{X}} \circ \pi_{\mathbb{X}_2} \quad j_1 \circ comp_{\mathbb{X}} = comp_{\mathbb{C}} \circ j_2$$

and the remaining equations $comp_{\mathbb{X}} \circ \langle i_{\mathbb{X}} \circ dom_{\mathbb{X}}, id_{\mathbb{X}_1} \rangle = id_{\mathbb{X}_1} = comp_{\mathbb{X}} \circ \langle id_{\mathbb{X}_1}, i_{\mathbb{X}} \circ cod_{\mathbb{X}} \rangle$ are also not difficult to prove.

We have seen that the pullback of an internal category \mathbb{C} along an arrow j with codomain \mathbb{C}_0 is an internal category:

► **Proposition 5.** Given an internal category \mathbb{C} and an arrow $j : X \rightarrow \mathbb{C}_0$ there is an internal category \mathbb{X} and an internal functor $\mathbb{j} : \mathbb{X} \rightarrow \mathbb{C}$ such that $\mathbb{X}_0 = X$ and $\mathbb{j}_0 = j$.

Moreover, this internal category \mathbb{X} , or rather $\mathbb{j} : \mathbb{X} \rightarrow \mathbb{C}$, has a universal property known as a cartesian lifting. To make this precise, we recall the notion of a fibred category, or fibration.

► **Definition 6** (Fibration [11, 20]). If $P : \mathcal{W} \rightarrow \mathcal{V}$ is a functor, then $\mathbb{j} : \mathbb{X} \rightarrow \mathbb{C}$ is a cartesian lifting of $j : X \rightarrow PC$ if for all $\mathbb{k} : \mathbb{W} \rightarrow \mathbb{C}$ and all $h : P\mathbb{W} \rightarrow X$ with $P\mathbb{k} = j \circ h$ there is a unique $\mathbb{h} : \mathbb{W} \rightarrow \mathbb{X}$ such that $\mathbb{j} \circ \mathbb{h} = \mathbb{k}$ and $P\mathbb{h} = h$. Moreover, $P : \mathcal{W} \rightarrow \mathcal{V}$ is called a (Grothendieck) fibration if all $j : X \rightarrow PC$ have a cartesian lifting for all \mathbb{C} in \mathcal{W} . If $P : \mathcal{W} \rightarrow \mathcal{V}$ is a fibration, the subcategory of \mathcal{W} that has as arrows the arrows \mathbb{f} such that $P\mathbb{f} = id_C$ is called the fibre over C .

The next lemma is a strengthening of Proposition 5.

► **Lemma 7.** Let \mathcal{V} be a category with finite limits. The forgetful functor $Cat(\mathcal{V}) \rightarrow \mathcal{V}$ is a fibration.

Instantiating Lemma 7 with $\mathbb{C} \times \mathbb{D}$ for \mathbb{C} and $j : \mathbb{C}_0 \otimes \mathbb{D}_0 \rightarrow \mathbb{C}_0 \times \mathbb{D}_0$ for $j : X_0 \rightarrow \mathbb{C}_0$, gives us the desired result that internal categories can be pulled back along arbitrary arrows between objects-of-objects:

► **Corollary 8.** The arrow $j : \mathbb{C}_0 \otimes \mathbb{D}_0 \rightarrow \mathbb{C}_0 \times \mathbb{D}_0$ lifts to a morphism of internal categories $\mathbb{j} : \mathbb{C} \otimes \mathbb{D} \rightarrow \mathbb{C} \times \mathbb{D}$. Moreover, \mathbb{j} is the cartesian lifting of j .

To show that this construction is functorial we need to use that $\otimes : \mathcal{V} \times \mathcal{V} \rightarrow \mathcal{V}$ is functorial and that $j : \mathbb{C}_0 \otimes \mathbb{D}_0 \rightarrow \mathbb{C}_0 \times \mathbb{D}_0$ is natural in \mathbb{C} and \mathbb{D} . In order to lift such natural transformations, which are arrows in the functor category $\mathcal{V}^{Cat(\mathcal{V}) \times Cat(\mathcal{V})}$, we use

► **Lemma 9.** If $P : \mathcal{E} \rightarrow \mathcal{B}$ is a fibration and \mathcal{A} is a category, then $P^{\mathcal{A}} : \mathcal{E}^{\mathcal{A}} \rightarrow \mathcal{B}^{\mathcal{A}}$ is a fibration.

Instantiating the lemma with $P = (-)_0 : \text{Cat}(\mathcal{V}) \rightarrow \mathcal{V}$ and $\mathcal{A} = \text{Cat}(\mathcal{V}) \times \text{Cat}(\mathcal{V})$, we obtain as a corollary that lifting the tensor $\otimes : \mathcal{V} \times \mathcal{V} \rightarrow \mathcal{V}$ to $\otimes : \text{Cat}(\mathcal{V}) \times \text{Cat}(\mathcal{V}) \rightarrow \text{Cat}(\mathcal{V})$ is functorial:

► **Theorem 10.** Let $(\mathcal{V}, 1, \otimes)$ be a (symmetric) monoidal category with finite limits in which the monoidal unit is the terminal object. Let $(-)_0 : \text{Cat}(\mathcal{V}) \rightarrow \mathcal{V}$ be the forgetful functor from categories internal in \mathcal{V} . Then the canonical arrow $j : \mathbb{C}_0 \otimes \mathbb{D}_0 \rightarrow \mathbb{C}_0 \times \mathbb{D}_0$ lifts to a natural transformation $\mathbb{j} : \mathbb{C} \otimes \mathbb{D} \rightarrow \mathbb{C} \times \mathbb{D}$. Moreover, $(\text{Cat}(\mathcal{V}), \mathbb{l}, \otimes)$ inherits from $(\mathcal{V}, 1, \otimes)$ the structure of a (symmetric) monoidal category with finite limits in which the monoidal unit is the terminal object.

In this paper we only need internal monoidal categories that are strict. In the same way as a strict monoidal category is a monoid in $(\text{Cat}, \mathbb{l}, \times)$, an internal strict monoidal category is a monoid in $(\text{Cat}(\mathcal{V}), \mathbb{l}, \otimes)$:

► **Definition 11 (Internal monoidal category).** Let $(\mathcal{V}, 1, \otimes)$ be a symmetric monoidal category with finite limits in which the monoidal unit is the terminal object and let $(\text{Cat}(\mathcal{V}), \mathbb{l}, \otimes)$ be the induced symmetric monoidal category of internal categories in \mathcal{V} . A strict internal monoidal category \mathbb{C} is a monoid $(\mathbb{C}, \emptyset, \odot)$ in $(\text{Cat}(\mathcal{V}), \mathbb{l}, \otimes)$.

More explicitly, a strict internal monoidal category \mathbb{C} has operations

$$\emptyset : \mathbb{l} \rightarrow \mathbb{C} \quad \odot : \mathbb{C} \otimes \mathbb{C} \rightarrow \mathbb{C}$$

satisfying the laws of a monoid. For example, in the category nF of finite sets of names, \emptyset is the empty set and $\uplus = \odot$ is, on objects, union of disjoint sets and, on arrows, union of functions with both disjoint domains and disjoint codomains. It follows from Remark 34 that an internal monoidal category satisfies the interchange law

$$\begin{array}{ccc} (\mathbb{C} \otimes \mathbb{C})_2 & \xrightarrow{\text{comp} \times \text{comp}} & (\mathbb{C} \otimes \mathbb{C})_1 \\ \downarrow \odot_2 & & \downarrow \odot_1 \\ \mathbb{C}_2 & \xrightarrow{\text{comp}} & \mathbb{C}_1 \end{array}$$

which can also be written as

$$(f \odot f'); (g \odot g') = (f; g) \odot (f'; g')$$

The move from \times to \otimes means that it is now possible that the right-hand side of the equation is defined while the left-hand side is not. But, as we will see, in nominal sets, the right-hand side is always α -equivalent to one for which a left-hand side exists.

4 Examples

Before we give a formal definition of nominal PROPs and nominal monoidal theories (NMTs) in the next section, we present as examples those NMTs that correspond to the SMTs of Fig 3. The significant differences between Fig 3 and 4 are that wires now carry labels and that there is a new generator $\frac{a_i}{\text{---}} \diamond \frac{b_i}{\text{---}}$ which allows us to change the label of a wire. Moreover, in the nominal setting rules for wire crossings are not needed.

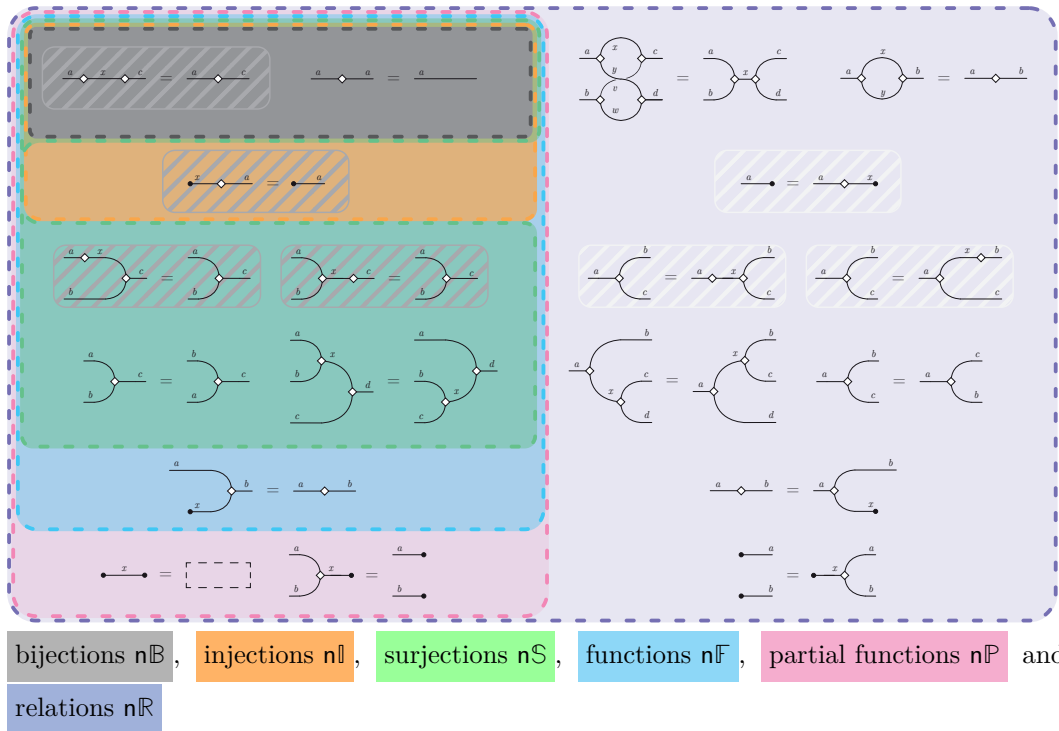


Figure 4 Nominal monoidal theories.

► **Theorem 12.** The calculi of Fig 4 are sound and complete, that is, the categories presented by these calculi are isomorphic the categories of finite sets of names with the respective maps.

The proof follows the same general lines as the well-known proofs for SMTs (see eg Lafont [14]) and proceed by showing that each diagram $f : A \rightarrow B$ can be rewritten to one in normal form, with the normal form being a direct syntactic representation of the semantic function/relation represented by f . The proofs for NMTs seem easier than the corresponding proofs for SMTs due to the absence of wire crossings. For example, in the case of bijections, it is immediate that, using the grey rules of Fig.4, every nominal diagram rewrites to a normal form which is just a parallel composition of diagrams of the form $\frac{a_i}{b_i}$.

5 Nominal monoidal theories and nominal PROPs

In this section, we introduce nominal PROPs as internal monoidal categories in nominal sets. We first spell out the details of what that means in elementary terms and then discuss the notion of diagrammatic alpha-equivalence.

5.1 Nominal monoidal theories

A *nominal monoidal theory* (Σ, E) is given by a nominal set Σ of generators and a nominal set E of equations. The set of nominal generators is itself generated by a set Σ_\circ of “ordinary” generators $\gamma : n \rightarrow m$, each γ giving rise to a set of nominal generators $[a]\gamma[b] : A \rightarrow B$ where a, b are unique lists of size n, m and whose underlying sets are A, B respectively. The

nominal generators Σ are closed under permutations

$$\pi \cdot [\mathbf{a}] \gamma \langle \mathbf{b} \rangle : \pi \cdot A \rightarrow \pi \cdot B = [\pi(\mathbf{a})] \gamma \langle \pi(\mathbf{b}) \rangle. \quad (\pi\text{-def})$$

The set of terms is given by closing under the operations of Fig 5, which should be compared with Fig 1.

| | | |
|--|--|---|
| $\frac{\gamma : m \rightarrow n \in \Sigma_o}{[\mathbf{a}] \gamma \langle \mathbf{b} \rangle : A \rightarrow B}$ | $\overline{id_a : \{a\} \rightarrow \{a\}}$ | $\overline{\delta_{ab} : \{a\} \rightarrow \{b\}}$ |
| $\frac{t : A \rightarrow B \quad t' : A' \rightarrow B'}{t \uplus t' : A \uplus A' \rightarrow B \uplus B'}$ | $\frac{t : A \rightarrow B \quad s : B \rightarrow C}{t; s : A \rightarrow C}$ | $\frac{t : A \rightarrow B}{(a \ b) t : (a \ b) \cdot A \rightarrow (a \ b) \cdot B}$ |

■ **Figure 5** NMT Terms.

Every NMT freely generates a monoidal category internal in nominal sets by quotienting the generated terms by:

- the equations that state that id and $;$ obey the laws of a category
- the equations stating that id_\emptyset and \uplus are a monoid
- the equations of an internal monoidal category of Fig 6 ³
- the equations of permutation actions of Fig 7
- the equations on the interaction of generators with bijections δ of Fig 8
- the equations E

| | |
|---|------------|
| $t \uplus s = s \uplus t$ | (NMT-comm) |
| $(s; t) \uplus (u; v) = (s \uplus u); (t \uplus v)$ | (NMT-ch) |

■ **Figure 6** NMT Equations of \uplus .

| | | |
|--|--|---|
| $(a \ b) id_x = id_{(a \ b) \cdot x}$ | $(a \ b) \delta_{xy} = \delta_{(a \ b) \cdot x \ (a \ b) \cdot y}$ | $(a \ b) \gamma = (a \ b) \cdot \gamma$ |
| $(a \ b)(x \uplus y) = (a \ b)x \uplus (a \ b)y$ | $(a \ b)(x; y) = (a \ b)x; (a \ b)y$ | |

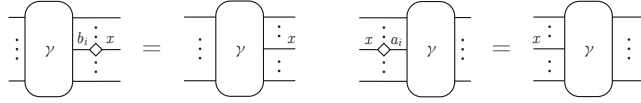
■ **Figure 7** NMT Equations of the permutation actions.


For terms to form a nominal set, we need equations between permutations (not listed here) to hold, as well as the equations of Fig 7 that specify how permutations act on terms. All the equations presented in the figures above are routine, with the possible exception of those of Fig 8, specifying the interaction of renamings δ with the generators $[\mathbf{a}] \gamma \langle \mathbf{b} \rangle \in \Sigma$, which we also depict in diagrammatic form:

³ The main difference with the equations in Fig 2 is that the interchange law for \uplus is required to hold only if both sides are defined and that the two laws involving symmetries are replaced by the commutativity of \uplus .

$$\begin{array}{c}
 \delta_{ab}; \delta_{bc} = \delta_{ac} \\
 \\
 \frac{[\mathbf{a}] \gamma \langle b_1, \dots, b_i, \dots, b_n \rangle : A \rightarrow B \uplus \{b_i\}}{[\mathbf{a}] \gamma \langle b_1, \dots, b_i, \dots, b_n \rangle ; (id_B \uplus \delta_{b_i x}) = [\mathbf{a}] \gamma \langle b_1, \dots, x, \dots, b_n \rangle} \quad \text{(NMT-right)} \\
 \\
 \frac{[a_1, \dots, a_i, \dots, a_m] \gamma \langle \mathbf{b} \rangle : \{a_i\} \uplus A \rightarrow B}{(\delta_{x a_i} \uplus id_A) ; [a_1, \dots, a_i, \dots, a_m] \gamma \langle \mathbf{b} \rangle = [a_1, \dots, x, \dots, a_m] \gamma \langle \mathbf{b} \rangle} \quad \text{(NMT-left)}
 \end{array}$$

■ **Figure 8** NMT Equations of renamings.



Instances of these rules can be seen in Fig 4, where they are distinguished by a  striped background.

5.2 Diagrammatic alpha-equivalence

The equations of Fig 7 and Fig 8 introduce a notion of *diagrammatic alpha-equivalence*, which allows us to rename “internal” names and to contract renamings.

► **Definition 13.** Two terms of a nominal monoidal theory are alpha-equivalent if their equality follows from the equations in Fig 7 and Fig 8.

Every permutation π of names gives rise to bijective functions $\pi_A : A \rightarrow \pi[A] = \{\pi(a) \mid a \in A\} = \pi \cdot A$. Any such π_A , as well as the inverse π_A^{-1} , are parallel compositions of δ_{ab} for suitable $a, b \in \mathcal{N}$. In fact, we have $\pi_A = \biguplus_{a \in A} \delta_{a \pi(a)}$. We may therefore use the π_A as abbreviations in terms.

► **Proposition 14.** Let $t : A \rightarrow B$ be a term of a nominal monoidal theory. The equations in Fig 7 and Fig 8 entail that $\pi \cdot t = (\pi_A)^{-1}; t; \pi_B$.

$$\begin{array}{ccc}
 A & \xrightarrow{t} & B \\
 \pi_A \downarrow & & \downarrow \pi_B \\
 \pi[A] & \xrightarrow{\pi \cdot t} & \pi[B]
 \end{array}$$

The next two corollaries show that internal names can be renamed. We call this diagrammatic α -equivalence.

► **Corollary 15.** Let $t : A \uplus \{c\} \rightarrow B \uplus \{c\}$ be a term of a nominal monoidal theory and d be fresh for t . Then $t = (\delta_{cd} \uplus id_A); (c d) \cdot t; (\delta_{dc} \uplus id_B)$.

► **Corollary 16.** Let $t : A \rightarrow B$ be a term of a nominal monoidal theory. Modulo the equations of Fig 7 and Fig 8, the support of t is $A \cup B$.

The last corollary shows that internal names are bound by sequential composition. Indeed, in a composition $A \xrightarrow{t} C \xrightarrow{s} B$, the names in $C \setminus (A \cup B)$ do not appear in the support of $t; s$.

5.3 Nominal PROPs

From the point of view of Section 3, a nominal PROP is an internal strict monoidal category in $(\mathbf{Nom}, 1, *)$ that has finite sets of names as objects and at least all bijections as arrows. A functor between nominal PROPs is an internal functor that preserves objects and bijections. We spell this out in detail.

► **Remark 17.** A nominal PROP \mathbb{C} is a small category, with a set \mathbb{C}_0 of “objects” and a set \mathbb{C}_1 of “arrows”, defined as follows. We write $;$ for the sequential composition (in the diagrammatic order) and \uplus for the monoidal composition.

- \mathbb{C}_0 is the set of finite subsets of a countably infinite set \mathcal{N} . The permutation action is given by $\pi \cdot A = \pi[A] = \{\pi(a) \mid a \in A\}$ for all finite permutations $\pi : \mathcal{N} \rightarrow \mathcal{N}$.
- \mathbb{C}_1 contains all bijections (“renamings”) $\pi_A : A \rightarrow \pi \cdot A$, $\pi_A(a) = \pi(a)$, for all finite permutations $\pi : \mathcal{N} \rightarrow \mathcal{N}$ and is closed under the operation mapping an arrow $f : A \rightarrow B$ to $\pi \cdot f : \pi \cdot A \rightarrow \pi \cdot B$ defined as $\pi \cdot f = (\pi_A)^{-1};f;\pi_B$.
- $A \uplus B$ is the union of A and B and defined whenever A and B are disjoint. This makes $(\mathbb{C}_0, \emptyset, \uplus)$ a commutative partial monoid. On arrows, we require $(\mathbb{C}_1, \emptyset, \uplus)$ to be a commutative partial monoid, with $f \uplus g$ defined whenever $\text{dom}f \cap \text{dom}g = \emptyset$ and $\text{cod}f \cap \text{cod}g = \emptyset$.
- The interchange law $(f \uplus f'); (g \uplus g') = (f;g) \uplus (f';g')$ holds whenever the left-hand side is defined.

From this definition one can deduce the following.

► **Remark 18.**

- A nominal PROP has a nominal set of objects and a nominal set of arrows.
- The support of an object A is A and the support of an arrow $f : A \rightarrow B$ is $A \cup B$. In particular, $\text{supp}(f;g) = \text{dom}(f) \cup \text{cod}(g)$. In other words, nominal PROPs have diagrammatic alpha equivalence.
- There is a category \mathbf{nPROP} that consists of nominal PROPs together with functors that are the identity on objects and bijections and are strict monoidal and equivariant.
- Every NMT presents a \mathbf{nPROP} . Conversely, every \mathbf{nPROP} is presented by at least one NMT given by all terms as generators and all equations.

6 Equivalence of nominal and ordinary string diagrams

We show that the categories \mathbf{nPROP} and \mathbf{PROP} are equivalent. To define translations between ordinary and nominal monoidal theories we introduce some auxiliary notation. We denote lists that contain each letter at most once by bold letters. If $\mathbf{a} = [a_1, \dots, a_n]$ is a list, then $\underline{\mathbf{a}} = \{a_1, \dots, a_n\}$. Given lists \mathbf{a} and \mathbf{a}' with $\underline{\mathbf{a}} = \underline{\mathbf{a}'}$ we abbreviate bijections in \mathbf{PROP} (also called symmetries) mapping $i \mapsto a_i = a'_j \mapsto j$ as $\langle \mathbf{a} | \mathbf{a}' \rangle$. Given lists \mathbf{a} and \mathbf{b} of the same length we write $[\mathbf{a} | \mathbf{b}] = \bigoplus \delta_{a_i b_i}$ for the bijection $a_i \mapsto b_i$ in an \mathbf{nPROP} .

► **Proposition 19.** For any PROP \mathcal{S} , there is an \mathbf{nPROP}

$$NOM(\mathcal{S})$$

that has for all arrows $f : \underline{n} \rightarrow \underline{m}$ of \mathcal{S} , and for all lists $\mathbf{a} = [a_1, \dots, a_n]$ and $\mathbf{b} = [b_1, \dots, b_m]$ arrows $[\mathbf{a}]f[\mathbf{b}] \in NOM(\mathcal{S})$. These arrows are subject to equations

18:14 Nominal String Diagrams

$$[\mathbf{a}]f; g\langle \mathbf{c} \rangle = [\mathbf{a}]f\langle \mathbf{b} \rangle; [\mathbf{b}]g\langle \mathbf{c} \rangle \quad (\text{NOM-1})$$

$$[\mathbf{a} \# \mathbf{c}]f \oplus g\langle \mathbf{b} \# \mathbf{d} \rangle = [\mathbf{a}]f\langle \mathbf{b} \rangle \uplus [\mathbf{c}]g\langle \mathbf{d} \rangle \quad (\text{NOM-2})$$

$$[\mathbf{a}]id\langle \mathbf{b} \rangle = [\mathbf{a}|\mathbf{b}] \quad (\text{NOM-3})$$

$$[\mathbf{a}] \langle \mathbf{b}|\mathbf{b}' \rangle; f \langle \mathbf{c} \rangle = [\mathbf{a}|\mathbf{b}]; [\mathbf{b}']f \langle \mathbf{c} \rangle \quad (\text{NOM-4})$$

$$[\mathbf{a}]f; \langle \mathbf{b}|\mathbf{b}' \rangle \langle \mathbf{c} \rangle = [\mathbf{a}]f\langle \mathbf{b} \rangle; [\mathbf{b}']\langle \mathbf{c} \rangle \quad (\text{NOM-5})$$

Proof. To show that $NOM(\mathcal{S})$ is well-defined, we need to check that the equations of \mathcal{S} are respected. We only have space here for the most interesting case which is the naturality of symmetries given by the last equation in Fig 2. We write \mathbf{a}^m for a list of \mathbf{a} 's of length m .

$$\begin{aligned} & [\mathbf{a}^m \# \mathbf{a}^z] (t \oplus id_z); \sigma_{n,z} \langle \mathbf{b}^z \# \mathbf{b}^n \rangle \\ &= ([\mathbf{a}^m] t \langle \mathbf{x}^n \rangle \uplus [\mathbf{a}^z] id_z \langle \mathbf{x}^z \rangle); [\mathbf{x}^n \# \mathbf{x}^z] \sigma_{n,z} \langle \mathbf{b}^z \# \mathbf{b}^n \rangle && (\text{NOM-1,2}) \\ &= ([\mathbf{a}^z] id_z \langle \mathbf{x}^z \rangle \uplus [\mathbf{a}^m] t \langle \mathbf{x}^n \rangle); [\mathbf{x}^n \# \mathbf{x}^z] \sigma_{n,z} \langle \mathbf{b}^z \# \mathbf{b}^n \rangle && (\text{NMT-comm}) \\ &= [\mathbf{a}^z \# \mathbf{a}^m] id_z \oplus t \langle \mathbf{x}^z \# \mathbf{x}^n \rangle; [\mathbf{x}^n \# \mathbf{x}^z] \sigma_{n,z} \langle \mathbf{b}^z \# \mathbf{b}^n \rangle && (\text{NOM-2}) \\ &= [\mathbf{a}^z \# \mathbf{a}^m] id_z \oplus t \langle \mathbf{x}^z \# \mathbf{x}^n \rangle; [\mathbf{x}^n \# \mathbf{x}^z] \langle \mathbf{x}^n \# \mathbf{x}^z | \mathbf{x}^z \# \mathbf{x}^n \rangle \langle \mathbf{b}^z \# \mathbf{b}^n \rangle && (\sigma\text{-def}) \\ &= [\mathbf{a}^z \# \mathbf{a}^m] id_z \oplus t \langle \mathbf{x}^z \# \mathbf{x}^n \rangle; [\mathbf{x}^n \# \mathbf{x}^z | \mathbf{x}^n \# \mathbf{x}^z]; [\mathbf{x}^z \# \mathbf{x}^n | \mathbf{b}^z \# \mathbf{b}^n] \\ &= [\mathbf{a}^z \# \mathbf{a}^m] id_z \oplus t \langle \mathbf{x}^z \# \mathbf{x}^n \rangle; [\mathbf{x}^z \# \mathbf{x}^n | \mathbf{b}^z \# \mathbf{b}^n] && (\delta_{aa} = id_a) \\ &= [\mathbf{a}^z \# \mathbf{a}^m] id_z \oplus t \langle \mathbf{b}^z \# \mathbf{b}^n \rangle && (\text{NOM-5}) \\ &= [\mathbf{a}^m \# \mathbf{a}^z | \mathbf{a}^m \# \mathbf{a}^z]; [\mathbf{a}^z \# \mathbf{a}^m] id_z \oplus t \langle \mathbf{b}^z \# \mathbf{b}^n \rangle && (\delta_{aa} = id_a) \\ &= [\mathbf{a}^m \# \mathbf{a}^z] \langle \mathbf{a}^m \# \mathbf{a}^z | \mathbf{a}^z \# \mathbf{a}^m \rangle; (id_z \oplus t) \langle \mathbf{b}^z \# \mathbf{b}^n \rangle && (\text{NOM-4}) \\ &= [\mathbf{a}^m \# \mathbf{a}^z] \sigma_{m,z}; (id_z \oplus t) \langle \mathbf{b}^z \# \mathbf{b}^n \rangle && (\sigma\text{-def}) \end{aligned}$$

Note how commutativity of \uplus is used to show that naturality of symmetries is respected. ◀

► **Proposition 20.** For any nPROP \mathcal{T} there is a PROP

$$ORD(\mathcal{T})$$

that has for all arrows $f : A \rightarrow B$ of \mathcal{T} , and for all lists $\mathbf{a} = [a_1, \dots, a_n]$ and $\mathbf{b} = [b_1, \dots, b_m]$ arrows $\langle \mathbf{a} \rangle f \langle \mathbf{b} \rangle$. These arrows are subject to equations

$$\langle \mathbf{a} \rangle f; g \langle \mathbf{c} \rangle = \langle \mathbf{a} \rangle f \langle \mathbf{b} \rangle; \langle \mathbf{b} \rangle g \langle \mathbf{c} \rangle \quad (\text{ORD-1})$$

$$\langle \mathbf{a}_f \# \mathbf{a}_g \rangle f \uplus g \langle \mathbf{b}_f \# \mathbf{b}_g \rangle = \langle \mathbf{a}_f \rangle f \langle \mathbf{b}_f \rangle \oplus \langle \mathbf{a}_g \rangle g \langle \mathbf{b}_g \rangle \quad (\text{ORD-2})$$

$$\langle \mathbf{a} \rangle id \langle \mathbf{a} \rangle = id \quad (\text{ORD-3})$$

$$\langle \mathbf{a} \rangle [\mathbf{a}'|\mathbf{b}]; f \langle \mathbf{c} \rangle = \langle \mathbf{a}|\mathbf{a}' \rangle; \langle \mathbf{b} \rangle f \langle \mathbf{c} \rangle \quad (\text{ORD-4})$$

$$\langle \mathbf{a} \rangle f; [\mathbf{b}|\mathbf{c}] \langle \mathbf{c}' \rangle = \langle \mathbf{a} \rangle f \langle \mathbf{b} \rangle; \langle \mathbf{c}|\mathbf{c}' \rangle \quad (\text{ORD-5})$$

Proof. To show that ORD is well-defined we need to show that the equations of an NMT are respected. The most interesting case here is the commutativity of \uplus since the \oplus of SMTs is not commutative.

$$\begin{aligned}
& \langle \mathbf{a}_t \# \mathbf{a}_s \rangle t \uplus s [\mathbf{b}_t \# \mathbf{b}_s] \\
&= \langle \mathbf{a}_t \rangle t [\mathbf{b}_t] \oplus \langle \mathbf{a}_s \rangle s [\mathbf{b}_s] && \text{(ORD-2)} \\
&= (\langle \mathbf{a}_t \rangle t [\mathbf{b}_t] ; id_{|\mathbf{b}_t|}) \oplus (id_{|\mathbf{a}_s|} ; \langle \mathbf{a}_s \rangle s [\mathbf{b}_s]) && (id; a = a = a; id) \\
&= (\langle \mathbf{a}_t \rangle t [\mathbf{b}_t] \oplus id_{|\mathbf{a}_s|}) ; (id_{|\mathbf{b}_t|} \oplus \langle \mathbf{a}_s \rangle s [\mathbf{b}_s]) && \text{(SMT-ch)} \\
&= (\langle \mathbf{a}_t \rangle t [\mathbf{b}_t] \oplus id_{|\mathbf{a}_s|}) ; \sigma_{|\mathbf{b}_t|, |\mathbf{a}_s|} ; \sigma_{|\mathbf{a}_s|, |\mathbf{b}_t|} ; (id_{|\mathbf{b}_t|} \oplus \langle \mathbf{a}_s \rangle s [\mathbf{b}_s]) && \text{(SMT-sym)} \\
&= \sigma_{|\mathbf{a}_t|, |\mathbf{a}_s|} ; (id_{|\mathbf{a}_s|} \oplus \langle \mathbf{a}_t \rangle t [\mathbf{b}_t]) ; \sigma_{|\mathbf{a}_s|, |\mathbf{b}_t|} ; (id_{|\mathbf{b}_t|} \oplus \langle \mathbf{a}_s \rangle s [\mathbf{b}_s]) && \text{(SMT-nat)} \\
&= \sigma_{|\mathbf{a}_t|, |\mathbf{a}_s|} ; (id_{|\mathbf{a}_s|} \oplus \langle \mathbf{a}_t \rangle t [\mathbf{b}_t]) ; (\langle \mathbf{a}_s \rangle s [\mathbf{b}_s] \oplus id_{|\mathbf{b}_t|}) ; \sigma_{|\mathbf{b}_s|, |\mathbf{b}_t|} && \text{(SMT-nat)} \\
&= \sigma_{|\mathbf{a}_t|, |\mathbf{a}_s|} ; ((id_{|\mathbf{a}_s|} ; \langle \mathbf{a}_s \rangle s [\mathbf{b}_s]) \oplus (\langle \mathbf{a}_t \rangle t [\mathbf{b}_t] ; id_{|\mathbf{b}_t|})) ; \sigma_{|\mathbf{b}_s|, |\mathbf{b}_t|} && \text{(SMT-ch)} \\
&= \sigma_{|\mathbf{a}_t|, |\mathbf{a}_s|} ; \langle \mathbf{a}_s \# \mathbf{a}_t \rangle s \uplus t [\mathbf{b}_s \# \mathbf{b}_t] ; \sigma_{|\mathbf{b}_s|, |\mathbf{b}_t|} && (id; a = a, \text{ORD-2}) \\
&= \langle \mathbf{a}_t \# \mathbf{a}_s | \mathbf{a}_s \# \mathbf{a}_t \rangle ; \langle \mathbf{a}_s \# \mathbf{a}_t \rangle s \uplus t [\mathbf{b}_s \# \mathbf{b}_t] ; \langle \mathbf{b}_s \# \mathbf{b}_t | \mathbf{b}_t \# \mathbf{b}_s \rangle && (\sigma\text{-def}) \\
&= \langle \mathbf{a}_t \# \mathbf{a}_s \rangle [\mathbf{a}_s \# \mathbf{a}_t | \mathbf{a}_s \# \mathbf{a}_t] ; s \uplus t ; [\mathbf{b}_s \# \mathbf{b}_t | \mathbf{b}_s \# \mathbf{b}_t] [\mathbf{b}_t \# \mathbf{b}_s] && \text{(ORD-4,5)} \\
&= \langle \mathbf{a}_t \# \mathbf{a}_s \rangle s \uplus t [\mathbf{b}_t \# \mathbf{b}_s] && (\delta_{aa} = id_a)
\end{aligned}$$

Note how naturality of symmetries is used to show that the definition of *ORD* respects commutativity of \uplus . \blacktriangleleft

Having described the maps *NOM* and *ORD* and shown they are homomorphisms, we now describe functors *NOM(F)* and *ORD(F)*.

► **Proposition 21.** *NOM* : PROP \rightarrow nPROP is a functor mapping an arrow of PROPs $F : \mathcal{S} \rightarrow \mathcal{S}$ to an arrow of nPROPs $NOM(F) : NOM(\mathcal{S}) \rightarrow NOM(\mathcal{S})$ defined by

$$NOM(F)(\langle \mathbf{a} \rangle g \langle \mathbf{b} \rangle) = \langle \mathbf{a} \rangle Fg \langle \mathbf{b} \rangle. \quad \text{(NOM-F)}$$

► **Proposition 22.** *ORD* is a functor mapping an arrow of nPROPs $F : \mathcal{T} \rightarrow \mathcal{T}$ to an arrow of PROPs $ORD(F) : ORD(\mathcal{T}) \rightarrow ORD(\mathcal{T})$ defined by

$$ORD(F)(\langle \mathbf{a} \rangle f \langle \mathbf{b} \rangle) = \langle \mathbf{a} \rangle Ff \langle \mathbf{b} \rangle \quad \text{(ORD-F)}$$

The next proposition has a variation in which we take PROPs in the weaker sense of Lack [13]. Then the unit $\mathcal{S} \rightarrow ORD(NOM(\mathcal{S}))$ is not an iso. To see where we need to be careful, the next example illustrates how the commutativity of \uplus in an nPROP translates into the naturality of the symmetries in a PROP.

► **Example 23** (Commutativity of \uplus translates to naturality of symmetries). If \mathcal{S} is a PROP in the sense of Lack [13] generated by a ‘‘lollipop’’ $\lambda : 0 \rightarrow 1$ then we can show that $\lambda \oplus id$ and $(id \oplus \lambda) ; \sigma_{1,1}$ in \mathcal{S} are sent to the same arrow in $ORD(NOM(\mathcal{S}))$, namely we can show $\langle a \rangle [a] \lambda \oplus id \langle b, c \rangle [b, c] = \langle a \rangle [a] (id \oplus \lambda) ; \sigma_{1,1} \langle b, c \rangle [b, c]$:

$$\begin{aligned}
\langle a \rangle [a] \lambda \oplus id \langle b, c \rangle [b, c] &= \langle a \rangle [a] \lambda \langle b \rangle \uplus [a] id \langle c \rangle [b, c] && \text{(NOM-2)} \\
&= \langle a \rangle [a] id \langle c \rangle \uplus [a] \lambda \langle b \rangle [b, c] && \text{(NMT-comm)} \\
&= \langle a \rangle [a] id \oplus \lambda \langle c, b \rangle [b, c] && \text{(NOM-2)} \\
&= \langle a \rangle [a] id \oplus \lambda \langle c, b \rangle ; [b, c | b, c] [b, c] && (a = a; id, \delta_{aa} = id_a) \\
&= \langle a \rangle [a] (id \oplus \lambda) ; \langle c, b | b, c \rangle \langle b, c \rangle [b, c] && \text{(NOM-5)} \\
&= \langle a \rangle [a] (id \oplus \lambda) ; \sigma_{1,1} \langle b, c \rangle [b, c] && (\sigma\text{-def})
\end{aligned}$$

which is an instance of (SMT-nat) and does not hold in \mathcal{S} .

18:16 Nominal String Diagrams

As we can see from the example, the naturality of symmetries in a PROP is necessary in order to obtain that $\mathcal{S} \rightarrow ORD(NOM(\mathcal{S}))$ is an iso in the next proposition.

► **Proposition 24.** For each PROP \mathcal{S} , there is an isomorphism of PROPs, natural in \mathcal{S} ,

$$\Delta : \mathcal{S} \rightarrow ORD(NOM(\mathcal{S}))$$

mapping $f \in \mathcal{S}$ to $\langle \mathbf{a} \rangle [\mathbf{a}] f \langle \mathbf{b} \rangle [\mathbf{b}]$ for some choice of \mathbf{a}, \mathbf{b} .

► **Proposition 25.** For each nPROP \mathcal{T} , there is an isomorphism of nPROPs, natural in \mathcal{T} ,

$$NOM(ORD(\mathcal{T})) \rightarrow \mathcal{T}$$

mapping the $[c] \langle \mathbf{a} \rangle f \langle \mathbf{b} \rangle [\mathbf{d}]$ generated by an $f : \underline{\mathbf{a}} \rightarrow \underline{\mathbf{b}}$ in \mathcal{T} to $[c|\mathbf{a}]; f; [\mathbf{b}|\mathbf{d}]$.

Since the last two propositions provide an isomorphic unit and counit of an adjunction, we obtain

► **Theorem 26.** The categories PROP and nPROP are equivalent.

► **Remark 27.** If we generalise the notion of PROP from MacLane [15] to Lack [13], in other words, if we drop equation (SMT-nat) of Fig 2 expressing the naturality of symmetries, we still obtain an adjunction, in which NOM is left-adjoint to ORD . Nominal PROPs then are a full reflective subcategory of ordinary PROPs. In other words, the (generalised) PROPs that satisfy naturality of symmetries are exactly those which are nominal PROPs.

7 Conclusion

The equivalence of nominal and ordinary PROPs (Theorem 26) has a satisfactory graphical interpretation. Indeed, comparing Figs 3 and 4 we see that both share, modulo different labellings of wires mediated by the functors ORD and NOM , the same core of generators and equations while the main difference lies in the equations expressing, on the one hand, that \oplus has natural symmetries and, on the other hand, that generators are a nominal set and \uplus is commutative. In fact, this can be taken as a justification of the importance of naturality, which, informally speaking, compensates for the irrelevant detail induced by ordering names.

There are several directions for future research. First, the notion of an internal monoidal category has been developed because it is easier to prove the basic results in general rather than only in the special case of nominal sets. Nevertheless, it would be interesting to explore whether there are other interesting instances of internal monoidal categories.

Second, internal monoidal categories are a principled way to build monoidal categories with a partial tensor. For example, by working internally in the category of nominal sets with the separated product we can capture in a natural way constraints such as the tensor $f \oplus g$ for two partial maps $f, g : \mathcal{N} \rightarrow V$ being defined only if the domains of f and g are disjoint. This reminds us of the work initiated by O’Hearn and Pym on categorical and algebraic models for separation logic and other resource logics, see eg [17, 9, 6]. It seems promising to investigate how to build categorical models for resource logics based on internal monoidal theories. In one direction, one could extend the work of Curien and Mimram [5] to partial monoidal categories. Another question is whether there is a more general strictification result characterising when a symmetric tensor can be replaced by a partial but commutative one.

Third, there has been substantial progress in exploiting Lack’s work on composing PROPs [13] in order to develop novel string diagrammatic calculi for a wide range of applications,

see eg [1, 2, 3, 22]. It will be interesting to explore how much of this technology can be transferred from PROPs to nominal PROPs.

Fourth, various applications of nominal string diagrams could be of interest. The original motivation for our work was to obtain a convenient calculus for simultaneous substitutions that can be integrated with multi-type display calculi [7] and, in particular, with the multi-type display calculus for first-order logic of Tzimoulis [21]. Another direction for applications comes from the work of Ghica and Lopez [10] on a nominal syntax for string diagrams. In particular, it would be of interest to add various binding operations to nominal PROPs.

References

- 1 John C. Baez, Brandon Coya, and Franciscus Rebro. Props in Network Theory. *Theory and Applications of Categories*, 33, 2018. URL: <http://www.tac.mta.ca/tac/volumes/33/25/33-25abs.html>.
- 2 Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Paweł Sobociński, and Fabio Zanasi. Rewriting Modulo Symmetric Monoidal Structure. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16*, pages 710–719, New York, NY, USA, 2016. ACM. doi:10.1145/2933575.2935316.
- 3 Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. The Calculus of Signal Flow Diagrams I. *Inf. Comput.*, 252(C):2–29, February 2017. doi:10.1016/j.ic.2016.03.002.
- 4 Bob Coecke and Aleks Kissinger. *Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning*. Cambridge University Press, 2017. doi:10.1017/9781316219317.
- 5 Pierre-Louis Curien and Samuel Mimram. Coherent Presentations of Monoidal Categories. *Logical Methods in Computer Science*, 13(3):1–38, September 2017. doi:10.23638/LMCS-13(3:31)2017.
- 6 Brijesh Dongol, Victor B. F. Gomes, and Georg Struth. A Program Construction and Verification Tool for Separation Logic. In *Mathematics of Program Construction - 12th International Conference, MPC 2015, Königswinter, Germany, June 29 - July 1, 2015. Proceedings*, pages 137–158, 2015. doi:10.1007/978-3-319-19797-5_7.
- 7 Sabine Frittella, Giuseppe Greco, Alexander Kurz, Alessandra Palmigiano, and Vlasta Sikimić. Multi-type display calculus for dynamic epistemic logic. *Journal of Logic and Computation*, 26(6):2017–2065, December 2014. doi:10.1093/logcom/exu068.
- 8 Murdoch J. Gabbay and Andrew M. Pitts. A New Approach to Abstract Syntax with Variable Binding. *Formal Aspects of Computing*, 13(3):341–363, July 2002. doi:10.1007/s001650200016.
- 9 D. Galmiche, D. Méry, and D. Pym. The Semantics of BI and Resource Tableaux. *Mathematical Structures in Comp. Sci.*, 15(6):1033–1088, December 2005. doi:10.1017/S0960129505004858.
- 10 Dan R. Ghica and Aliaume Lopez. A structural and nominal syntax for diagrams. In *Proceedings 14th International Conference on Quantum Physics and Logic, QPL 2017, Nijmegen, The Netherlands, 3-7 July 2017.*, pages 71–83, 2017. doi:10.4204/EPTCS.266.4.
- 11 Bart Jacobs. *Categorical Logic and Type Theory*. Studies in logic and the foundations of mathematics. Elsevier Science, 1999.
- 12 André Joyal and Ross Street. The geometry of tensor calculus, I. *Advances in Mathematics*, 88(1):55–112, 1991. doi:10.1016/0001-8708(91)90003-P.
- 13 Steve Lack. Composing PROPs. *Theory and Applications of Categories*, 13:147–163, 2004. URL: <http://www.tac.mta.ca/tac/volumes/13/9/13-09abs.html>.
- 14 Yves Lafont. Towards an algebraic theory of Boolean circuits. *Journal of Pure and Applied Algebra*, 184:257–310, 2003. doi:10.1016/S0022-4049(03)00069-0.
- 15 Saunders Mac Lane. *Categories for the Working Mathematician*. Springer New York, 1978. doi:10.1007/978-1-4757-4721-8.

16 Saunders MacLane. Categorical algebra. *Bull. Amer. Math. Soc.*, 71(1):40–106, January 1965. doi:10.1090/S0002-9904-1965-11234-4.

17 Peter W. O’Hearn and David J. Pym. The Logic of Bunched Implications. *The Bulletin of Symbolic Logic*, 5(2):215–244, 1999. doi:10.2307/421090.

18 Andrew M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2013. doi:10.1017/CB09781139084673.

19 P. Selinger. *A Survey of Graphical Languages for Monoidal Categories*, pages 289–355. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. doi:10.1007/978-3-642-12821-9_4.

20 Thomas Streicher. Fibred Categories à la Jean Bénabou, 1999. arXiv:arXiv:1801.02927.

21 Apostolos Tzimoulis. *Algebraic and Proof-Theoretic Foundations of the Logics for Social Behaviour*. PhD thesis, Delft University of Technology, 2018. doi:10.4233/uuid:e67e7724-b378-4ca3-ad4e-c40df245af5e.

22 Fabio Zanasi. *Interacting Hopf Algebras- the Theory of Linear Systems*. Theses, Ecole normale supérieure de lyon - ENS LYON, October 2015. URL: <https://tel.archives-ouvertes.fr/tel-01218015>.

A Some internal category theory

See eg Borceux, Handbook of Categorical Algebra, Volume 1, Chapter 8 and the nlab.

► **Definition 28** (internal category). *In a category with finite limits an internal category is a diagram*

$$\begin{array}{ccccc}
 & \xrightarrow{\text{right}} & & \xrightarrow{\pi_2} & & \xrightarrow{\text{dom}} & \\
 A_3 & \xrightarrow{\text{compr}} & A_2 & \xrightarrow{\text{comp}} & A_1 & \xleftarrow{i} & A_0 \\
 & \xrightarrow{\text{compl}} & & \xrightarrow{\pi_1} & & \xrightarrow{\text{cod}} & \\
 & \xrightarrow{\text{left}} & & & & &
 \end{array} \tag{5}$$

such that

1. the “pairs of arrows”-object A_2 $\begin{array}{ccc} A_2 & \xrightarrow{\pi_2} & A_1 \\ \pi_1 \downarrow & & \downarrow \text{dom} \\ A_1 & \xrightarrow{\text{cod}} & A_0 \end{array}$ is a pullback,
2. the “triple of arrows”-object A_3 is a pullback

$$\begin{array}{ccc}
 A_3 & \xrightarrow{\text{right}} & A_2 \\
 \text{left} \downarrow & & \downarrow \pi_1 \\
 A_2 & \xrightarrow{\pi_2} & A_1
 \end{array}$$

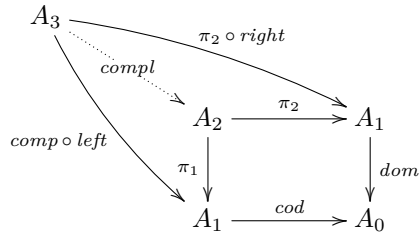
where, intuitively, left “projects out the left two arrows” and right “projects out the right two arrows”

3. $\text{dom} \circ \text{comp} = \text{dom} \circ \pi_1$ and $\text{cod} \circ \text{comp} = \text{cod} \circ \pi_2$,
4. $\text{dom} \circ i = \text{id}_{A_0} = \text{cod} \circ i$,
5. $\text{comp} \circ \langle i \circ \text{dom}, \text{id}_{A_1} \rangle = \text{id}_{A_1} = \text{comp} \circ \langle \text{id}_{A_1}, i \circ \text{cod} \rangle$
6. $\text{comp} \circ \text{compl} = \text{comp} \circ \text{compr}$

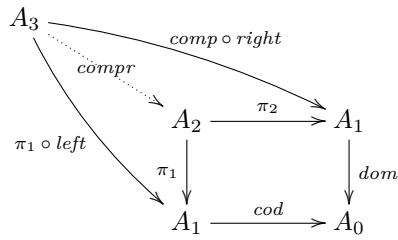
where we use the auxiliary notation

- $\langle i \circ \text{dom}, \text{id}_{A_1} \rangle : A_1 \rightarrow A_2$ and $\langle \text{id}_{A_1}, i \circ \text{cod} \rangle : A_1 \rightarrow A_2$ are the arrows into the pullback A_2 pairing $i \circ \text{dom}, \text{id}_{A_1} : A_1 \rightarrow A_1$ and $\text{id}_{A_1}, i \circ \text{cod} : A_1 \rightarrow A_1$, respectively,

- *compl* is the arrow composing the “left two arrows”



- *compr* is the arrow composing the “right two arrows”



► Remark 29. 1. and 3. define A_2 as the “object of composable pairs of arrows” while 4. and 5. express that the “object of arrows” A_1 has identities. 2. and 5. formalise associativity of composition. Since A_2 and A_3 are pullbacks, the structure is determined by $(A_0, A_1, \text{dom}, \text{cod}, i, \text{comp})$ alone. We included A_2, A_3 as well as *compr*, *compl*, *right*, *left*, π_2, π_1 to improve readability of the equations.

► Definition 30. A morphism $f : A \rightarrow B$ between internal categories, an internal functor, is a pair (f_0, f_1) of arrows such that the six squares (one for each of $\pi_2, \text{comp}, \pi_1, \text{dom}, \text{cod}, i$)

$$\begin{array}{ccccc}
 & \xrightarrow{\pi_2} & & \xrightarrow{\text{dom}} & \\
 A_2 & \xrightarrow{\text{comp}} & A_1 & \xleftarrow{i} & A_0 \\
 & \xrightarrow{\pi_1} & & \xrightarrow{\text{cod}} & \\
 \downarrow f_2 & & \downarrow f_1 & & \downarrow f_0 \\
 B_2 & \xrightarrow{\pi_2} & B_1 & \xrightarrow{\text{dom}} & B_0 \\
 & \xrightarrow{\text{comp}} & & \xleftarrow{i} & \\
 & \xrightarrow{\pi_1} & & \xrightarrow{\text{cod}} &
 \end{array} \tag{6}$$

commute.

- Remark 31. ■ Because B_2 is a pullback f_2 is uniquely determined by f_1 . In more detail, if $\Gamma \rightarrow B_2$ is any arrow then, because B_2 is a pullback, it can be written as a pair

$$\langle l, r \rangle : \Gamma \rightarrow B_2 \tag{7}$$

of arrows $l, r : \Gamma \rightarrow B_1$ and f_2 is determined by f_1 via

$$f_2 \circ \langle l, r \rangle = \langle f_1 \circ l, f_1 \circ r \rangle \tag{8}$$

- Even if f_2 is not needed as part of the structure in the above definition, including f_2 makes it easier to state that f_1 preserves composition.
- Similarly, B_3 is a pullback, and there is a unique arrow f_3 such that (f_0, f_1, f_2, f_3) together make further 4 squares commute, one for each of *right*, *compr*, *compl*, *left*, see (5). We may include f_3 in the structure whenever convenient.

18:20 Nominal String Diagrams

► **Definition 32.** A natural transformation $\alpha : f \rightarrow g$ between internal functors $f, g : A \rightarrow B$, an internal natural transformation, is an arrow $\alpha : A_0 \rightarrow B_1$ such that, recalling (7),

$$\text{dom} \circ \alpha = f_0 \quad \text{cod} \circ \alpha = g_0 \quad \text{comp} \circ \langle f_1, \alpha \circ \text{cod} \rangle = \text{comp} \circ \langle \alpha \circ \text{dom}, g_1 \rangle$$

► **Remark 33.** Internal categories with functors and natural transformations form a 2-category. We denote by $\text{Cat}(\mathcal{V})$ the category or 2-category of categories internal in \mathcal{V} . The forgetful functor $\text{Cat}(\mathcal{V}) \rightarrow \mathcal{C}$ mapping an internal category A to its object of objects A_0 has both left and right adjoints and, therefore, preserves limits and colimits. Moreover, a limit of internal categories is computed component-wise as $(\lim D)_j = \lim(D_j)$ for $j = 0, 1, 2$.

► **Remark 34.** A strict monoidal category can be thought of both as a monoid in the category of categories and as a category internal in the category of monoids. To understand this in more detail, note that both cases give rise to the diagram

$$\begin{array}{ccccc}
 A_2 \times A_2 & \xrightarrow{\text{comp} \times \text{comp}} & A_1 \times A_1 & \xrightarrow[\text{cod} \times \text{cod}]{\text{dom} \times \text{dom}} & A_0 \times A_0 \\
 \downarrow m_2 & & \downarrow m_1 & & \downarrow m_0 \\
 A_2 & \xrightarrow{\text{comp}} & A_1 & \xrightarrow[\text{cod}]{\text{dom}} & A_0
 \end{array}$$

where

- in the case of a monoid A in the category of internal categories, $m = (m_0, m_1, m_2)$ is an internal functor $A \times A \rightarrow A$ and, using that products of internal categories are computed component-wise, we have $\text{comp} \circ m_2 = m_1 \circ (\text{comp} \times \text{comp})$, which gives us the interchange law

$$(f; g) \cdot (f'; g') = (f \cdot f') ; (g \cdot g')$$

by using (8) with m for f and writing $;$ for comp and \cdot for m_1 ;

- in the case of a category internal in monoids we have monoids A_0, A_1, A_2 and monoid homomorphisms $i, \text{dom}, \text{cod}, \text{comp}$ which, if spelled out, leads to the same commuting diagrams as the previous item.

A Diagrammatic Approach to Quantum Dynamics

Stefano Gogioso 

University of Oxford, UK

stefano.gogioso@cs.ox.ac.uk

Abstract

We present a diagrammatic approach to quantum dynamics based on the categorical algebraic structure of strongly complementary observables. We provide physical semantics to our approach in terms of quantum clocks and quantisation of time. We show that quantum dynamical systems arise naturally as the algebras of a certain dagger Frobenius monad, with the morphisms and tensor product of the category of algebras playing the role, respectively, of equivariant transformations and synchronised parallel composition of dynamical systems. We show that the Weyl Canonical Commutation Relations between time and energy are an incarnation of the bialgebra law and we derive Schrödinger's equation from a process-theoretic perspective. Finally, we use diagrammatic symmetry-observable duality to prove Stone's proposition and von Neumann's Mean Ergodic proposition, recasting the results as two faces of the very same coin.

2012 ACM Subject Classification Theory of computation → Quantum computation theory; Theory of computation → Categorical semantics

Keywords and phrases Quantum dynamics, String diagrams, Categorical algebra

Digital Object Identifier 10.4230/LIPIcs.CALCO.2019.19

Acknowledgements This publication was made possible through the support of a grant from the John Templeton Foundation. The opinions expressed in this publication are those of the authors and do not necessarily reflect the views of the John Templeton Foundation.

1 Introduction

It is hard to overstate the importance of quantum dynamics: in a very deep sense, it truly makes the world go round. In computer science, more specifically, it is the driving force behind the processes which underpin the entirety of quantum computation. Despite this crucial role, quantum dynamics is rarely considered directly in quantum information and in the foundations of quantum computing, being instead relegated to a lower level of abstraction. In this work we aim to change that, bringing dynamics on a par with information and circuits by developing a fully diagrammatic approach based on categorical algebra.

Our work fits within the framework of categorical quantum mechanics [1, 2, 7] and uses the graphical calculus of string diagrams for symmetric monoidal categories [20, 26]. We consider a particularly well-behaved kind of Hopf algebras/bialgebras – closely related to compact quantum groups [30, 31] and sometimes known as *interacting quantum observables* [5, 10] – arising as strongly complementary pairs of symmetric \dagger -Frobenius algebras [9, 27]. We show that the algebras of a certain dagger Frobenius monad [18] correspond to quantum dynamical systems, that the morphisms between algebras correspond to equivariant transformations and that the natural tensor product in the category of algebras corresponds to synchronised parallel composition of dynamical systems. We further show that the dagger monoidal structure corresponds to symmetry-observable duality between time and energy, so that the Hamiltonian observables for quantum dynamical systems arise as the coalgebras of the corresponding dagger Frobenius comonad.



© Stefano Gogioso;

licensed under Creative Commons License CC-BY

8th Conference on Algebra and Coalgebra in Computer Science (CALCO 2019).

Editors: Markus Roggenbach and Ana Sokolova; Article No. 19; pp. 19:1–19:23

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

To showcase the expressive power of our formalism, we derive some cornerstone results of quantum dynamics in a diagrammatic fashion: the Weyl Canonical Commutation Relations, Schrödinger’s Equation, Stone’s proposition and von Neumann’s Mean Ergodic proposition. We use non-standard analysis [14, 15] to deal with infinite-dimensional quantum systems.

2 Interacting Quantum Observables

2.1 Quantum Observables

If \mathcal{G} is an object in a dagger symmetric monoidal category, a \dagger -Frobenius algebra \bullet on \mathcal{G} is a pair of a monoid (\mathcal{G}, μ, ν) and a comonoid $(\mathcal{G}, \nu^\dagger, \mu^\dagger) := (\mathcal{G}, \mu^\dagger, \nu^\dagger)$ related by *Frobenius law*:

$$(1)$$

A \dagger -Frobenius algebra is *quasi-special* if the comultiplication is an isometry up to some invertible scalar ξ and it is *special* if it is an actual isometry (i.e. $\xi = 1$):

$$(2)$$

The positive scalar $N_\bullet := \xi^\dagger \xi$ is known as the *normalisation factor* for the algebra. Every quasi-special \dagger -Frobenius algebra is proportional to a special one, which we shall see shortly to have physical significance: we use quasi-special algebras merely for reasons of notational convenience. The cup and cap induced by a \dagger -Frobenius algebra satisfy the *snake equation*:

$$(3)$$

A \dagger -Frobenius algebra is *symmetric* if the cup and cap it induces are symmetric:

$$(4)$$

Special symmetric \dagger -Frobenius algebras are of fundamental importance to quantum information, because they correspond exactly to quantum observables (more precisely, to finite-dimensional C^* -algebras [27]). Special *commutative* \dagger -Frobenius algebras in particular correspond to non-degenerate quantum observables, i.e. orthonormal bases. The basis vectors are exactly the *classical states* $|g\rangle$ for the algebra:

$$(5)$$

The leftmost two equations are the *copying* and *deleting* of the classical elements, while the rightmost equation says that classical elements are self-conjugate with respect to the

19:4 A Diagrammatic Approach to Quantum Dynamics

Then antipode always corresponds to the group inverse:

$$\begin{array}{c} \square \\ | \\ \textcircled{g} \end{array} = \begin{array}{c} | \\ \textcircled{g^{-1}} \end{array} \tag{10}$$

The classical states $|\chi\rangle$ for G are exactly those in the following form, where $\chi : G \rightarrow \mathbb{C}$ is a multiplicative character for G :

$$|\chi\rangle := \sum_{g \in G} \chi(g) |g\rangle \tag{11}$$

If G is abelian, then the pair $(\mathcal{G}, \bullet, \circ)$ corresponds to the Pontryagin dual group G^\wedge .

By thinking of the comonoid (φ, ψ) as copying/deleting group elements and of the monoid $(\blacktriangle, \bullet)$ as a ‘‘coherent’’/linear version of the group multiplication and unit, the defining equations for interacting quantum observables acquire a rather obvious meaning:

1. the bialgebra law and left coherence law say that group multiplication sends elements which can be copied/deleted to elements which can be copied/deleted;
2. the right coherence law and the bone law say that the group unit is itself and element which can be copied/deleted;
3. the requirement that the antipode is self-inverse is necessary to prove Hopf’s law, which in turn implies that elements have inverses (given by the antipode itself);
4. the requirement that the antipode is self-inverse also implies that group multiplication sends self-conjugate elements for \circ to self-conjugate elements for \bullet and the the group unit is itself self-conjugate for \circ [10, 16].

This means that we can think of interacting quantum observables as ‘‘coherent groups’’, i.e. groups embedded into an environment in which \dagger -Frobenius algebras are available (e.g. in the compact-closed complex linear context). This is consistent with the fact that such coherent groups are particularly well-behaved examples of compact quantum groups [13, 30, 31] – at least within the context of dagger compact categories of finite-dimensional vectors spaces over a field equipped with a self-inverse automorphism (acting as *conjugation*).

2.3 Dagger Frobenius Monads

Given any monoid $(\mathcal{G}, \blacktriangle, \bullet)$ in a monoidal category, we can always define a monad by sending $\mathcal{H} \mapsto \mathcal{H} \otimes \mathcal{G}$, $f \mapsto f \otimes \text{id}_{\mathcal{G}}$ and considering the following multiplication and unit:

$$\mu_{\mathcal{H}} := \begin{array}{c} \mathcal{H} \quad \mathcal{G} \\ | \quad \bullet \\ \mathcal{H} \quad \mathcal{G} \quad \mathcal{G} \end{array} \qquad \eta_{\mathcal{H}} := \begin{array}{c} \mathcal{H} \quad \mathcal{G} \\ | \quad \bullet \\ \mathcal{H} \end{array} \tag{12}$$

The monad laws reduce to associativity and bilateral unitality for the monoid itself:

$$\begin{array}{c} | \\ \bullet \\ | \end{array} = \begin{array}{c} | \\ \bullet \\ | \end{array} \qquad \begin{array}{c} | \\ \bullet \\ | \end{array} = \begin{array}{c} | \\ | \\ | \end{array} \qquad \begin{array}{c} | \\ \bullet \\ | \end{array} = \begin{array}{c} | \\ | \\ | \end{array} \tag{13}$$

If the category is dagger monoidal and the monoid is part of a \dagger -Frobenius algebra, then the monad is in fact a *dagger Frobenius monad* [18], i.e. it satisfies the following law relating it

to the comonad induced by the comonoid $(\mathcal{G}, \forall, \circ)$:

$$(14)$$

Because no ambiguity can arise, we write $_ \otimes \bullet$ to denote both the monad given by $(\mathcal{G}, \bullet, \circ)$ and the comonad given by $(\mathcal{G}, \forall, \circ)$. The *algebras* for the dagger Frobenius monad $_ \otimes \bullet$ are the morphisms $\alpha : \mathcal{H} \otimes \mathcal{G} \rightarrow \mathcal{H}$ such that:

$$(15)$$

In the original [18], these are referred to more specifically as *FEM-algebras*, for *Frobenius Eilenberg-Moore algebras*. Just as the “Eilenberg-Moore/EM” qualifier is customarily dropped for algebras of a monad, we shall here also drop the “Frobenius/FEM” qualifier for the algebras of dagger Frobenius monads, because FEM-algebras are the natural notion in the dagger Frobenius context.

► **Proposition 3.** *When $(\mathcal{G}, \bullet, \circ)$ is a pair of interacting quantum observables, the right-most condition above in the definition of algebras for the monad $_ \otimes \bullet$ can be equivalently reformulated as follows:*

$$(16)$$

3 Quantum Clocks

In a very practical sense, time is ticked by clocks. If we know that a dynamical system and a clock are synchronised, then we can know the exact state of the system without ever looking at it, by just knowing what time the clock is displaying (assuming we know the initial state for the system). The kinds of dynamics admissible for systems synchronised with a given clock depend on the structure of the clock: clocks with more time states can be synchronised with more systems. If one interprets dynamics as time-translation symmetry – the approach that we take in this work – this means that the dynamical system has to be a representation for whatever time-translation group is associated with the clock.

The interpretation of dynamics as time-translation symmetry may appear causally problematic: after all, what does it mean to have entanglement across time? A full discussion of this issue would take more words than can fit in the margins of these pages, but we hope the following will at least convince the reader that the point of view we’ve adopted might not be as preposterous as it may at first seem.

When thinking about time, we can take two perspectives: an *internal* perspective – reflecting time as *dynamically* experienced by those immersed in its flow – and an *external* perspective – reflecting time as *statically* experienced by those staring at it from the outside.

Mathematically, the internal perspective roughly corresponds to the idea of time evolution as the solution of differential equations: an instantaneous state is given and propagated forward instant by instant according to the laws of dynamics. The external perspective, on the other hand, corresponds to the idea of spacetime: everything which happens in the entire history of a mathematical objects is already crystallised in front of the eyes of those studying it, its evolution merely a matter of choosing and relating equal-time slices.

In the case of quantum dynamics, the external perspective amounts to thinking of dynamical systems as static, their entire history of evolution $(|\psi_t\rangle)_t$ encoded in an entangled state $\sum_t |\psi_t\rangle \otimes |t\rangle$, where we selected a suitably large ancillary quantum system \mathcal{G} – a *quantum clock*, as we shall call it – equipped with a choice of *time observable* \odot labelling the time states $|t\rangle$. This idea was already clear in the formulation of Feynman’s clock [11,12]: therein, the computation of the dynamics of a quantum system is reduced to the computation of the ground state of a certain Hamiltonian, resulting precisely in the system-clock entangled state detailed above. Causality and dynamics arise from the choice of a specific group structure on the time states – the *time-translation group* – corresponding to a strongly complementary quantum observable \bullet : the time observable chooses the time slices for the dynamical system, the group structure relates them causally and dynamically.

For an internal observer, living inside the quantum dynamical system, the classical evolution $(|\psi_t\rangle)_t$ is indistinguishable from the external application of time-translation symmetry to slices of definite time value $|t\rangle$: to such an internal observer, time behaves as an external classical parameter. To an external observer, on the other hand, the difference between the two perspectives is truly one of expressive power: taking the internal perspective forces them to work in the time observable for the quantum clock, while taking the external perspective allows the to freely change their point of view, yielding additional insights and more direct proofs of canonical results.

3.1 Quantum clocks from interacting quantum observables – Take I

Here, we restrict our attention to four inter-related kinds of dynamics: discrete periodic, discrete, continuous periodic and continuous.

1. Systems with continuous dynamics correspond to the the usual choice of time-translation group \mathbb{R} (continuous time).
2. Systems with continuous periodic dynamics correspond to a choice of time-translation group in the form $\mathbb{R}/T\mathbb{Z}$ for some positive period $T \in \mathbb{R}$. They are exactly the systems with continuous dynamics where the dynamics are periodic.
3. Systems with discrete dynamics correspond to the choice of time-translation group \mathbb{Z} . Sampling systems with continuous dynamics at equally spaced discrete intervals of time yields systems with discrete dynamics. These are the systems which are effectively synchronised with ordinary clocks (as long as we assume access to infinite time counters which never cycle).
4. Systems with discrete periodic dynamics correspond to a choice of time-translation group in the form \mathbb{Z}_N for some positive period $N \in \mathbb{N}$. Sampling systems with continuous periodic dynamics (with period $T \in \mathbb{R}$) at equally spaced discrete intervals of time (spaced by some positive $\Delta t \in \mathbb{R}$ dividing the period T) yields systems with discrete dynamics (with $N := T/\Delta t$). These are the systems which are effectively synchronised with ordinary clocks having finite time counters – from 12-hour wall clocks, with their 43200 time states, to high-precision atomic clocks.

When G is one of the time-translation groups above, a dynamical system *governed by* G – i.e. one which can be synchronised with clocks having G as the associated time-translation group – is simply a representation $(U_g)_{g \in G}$ of G in some appropriate category modelling the physical

context of interest. In particular, a quantum dynamical system is just a unitary representation $(U_g)_{g \in G}$ of G on some Hilbert space \mathcal{H} , with appropriate continuity requirements imposed where necessary.

The identification of quantum dynamical systems with unitary representations is the mainstream view in quantum dynamics, but it introduces an unpleasant asymmetry between the physical systems – which are quantum – and time – which is instead a classical parameter external to the quantum realm. This asymmetry did not escape the attention of the founders of quantum mechanics, and the history of attempts to quantise time is long and rife with controversy. We refer the reader interested in such history to some very good works dedicated specifically to the topic [4, 19, 23, 24]: in this work, we will instead avoid such controversies altogether, by taking an external “static” view of quantum dynamical systems.

For the purposes of dynamics, we have seen that a clock – a physical system – can be abstracted to a time-translation group G – an algebraic structure that it can be endowed with. If we take an ordinary clock – the states of which are the possible instants of the time ticks – and we quantise it, we obtain a *quantum clock* – the states of which are now *wavefunctions* over the space of states for the original clock. In this interpretation, quantum clocks should be quantum systems equipped with the structure of a group algebra $\mathbb{C}[G]$ for the time-translation group G .

In the case $G = \mathbb{Z}_N$ of discrete periodic dynamics we already know what to do: a quantum clock is finite-dimensional Hilbert space \mathcal{G} – a quantum system, living in the dagger compact category \mathbf{fHilb} – endowed with a pair $(\mathcal{G}, \bullet, \circ)$ of interacting quantum observables corresponding to $\mathbb{C}[\mathbb{Z}_N]$ – a pair of categorical algebraic structures. Unlike classical clocks – where a single algebraic object was needed – quantum clocks need an interacting *pair* of algebraic structures: one to pin down the time states (the observable \bullet) and another one to endow them with the \mathbb{Z}_N group structure (the observable \circ).

3.2 Infinite-dimensional quantum systems

This approach – modelling quantum clocks using interacting quantum observables – will work well for finite-dimensional quantum dynamical systems with discrete periodic dynamics, which are by themselves of significant interest: they were extensively studied by Weyl [28, 29] and can be used to formalise Feynman’s clock construction [11, 12, 22]. However, it cannot immediately be generalised to the other kinds of dynamics which we are interested in: the only pairs of interacting quantum observables in the dagger symmetric monoidal category \mathbf{Hilb} of Hilbert spaces and continuous linear maps are the ones corresponding to finite groups. Technically, \mathbf{Hilb} doesn’t even have quantum observables as we defined them: an orthonormal basis $(|e_i\rangle)_{i=1}^{\infty}$ of an infinite-dimensional separable Hilbert space is still associated with pair of a commutative comultiplication $\varphi := \sum_{i=1}^{\infty} (|e_i\rangle \otimes |e_i\rangle) \langle e_i|$ and multiplication $\psi = \varphi^\dagger$ satisfying the Frobenius law – as well as an alternative equation making classical states self-conjugate – but we have to let go of the counit η and unit ϵ in the passage from finite- to infinite-dimensional Hilbert spaces [3]. Indeed, the unit for such an algebra would have to take the form $\epsilon = \sum_{i=1}^{\infty} |e_i\rangle$, a vector which would have infinite norm.

In order to gain access to interacting quantum observables corresponding to infinite group algebras, we work in a symmetric monoidal category of non-standard Hilbert spaces and ${}^*\mathbb{C}$ -linear maps known as ${}^*\mathbf{Hilb}$ [14, 15], where ${}^*\mathbb{C}$ is the field of non-standard complex numbers. The objects of ${}^*\mathbf{Hilb}$ are non-standard Hilbert spaces which are *hyperfinite*-dimensional, i.e. which have orthonormal bases in the form $(|e_i\rangle)_{i=1}^n$ where $n \in {}^*\mathbb{N}$ is a non-standard natural number (the *dimension* of the non-standard Hilbert space); in particular, ${}^*\mathbf{Hilb}$ contains symmetric monoidal sub-categories equivalent to \mathbf{fHilb} and \mathbf{Hilb} (up to infinitesimals). Even

though n may be an infinite natural, from a non-standard perspective the objects of ${}^*\text{Hilb}$ are finite-dimensional spaces: this means that ${}^*\text{Hilb}$ is dagger compact, has special symmetric \dagger -Frobenius algebras and contains the extra strongly complementary pairs which we need to talk about quantum dynamics.

The easiest way to work with ${}^*\text{Hilb}$ is by using the *Transfer Principle*: if a construction indexed by n can be made on n -dimensional Hilbert spaces for all $n \in \mathbb{N}$, then it can be uniquely extended to n -dimensional non-standard Hilbert spaces for all $n \in {}^*\mathbb{N}$. For an extensive introduction to non-standard analysis and the Transfer Principle we refer the reader to Refs. [17, 25]. For example, if $(|e_i\rangle)_{i=1}^n$ is an orthonormal basis for an n -dimensional Hilbert space, we can always define the unit for the associated quantum observable as $\phi = \sum_{i=1}^n |e_i\rangle$: by the Transfer Principle, this means that we can also do so for an orthonormal basis $(|e_i\rangle)_{i=1}^n$ of an object of ${}^*\text{Hilb}$ where n is an infinite natural. The vector $\phi = \sum_{i=1}^n |e_i\rangle$ has a well-defined infinite square norm $\sum_{i=1}^n \langle e_i | e_i \rangle = n$ and can be normalised to $\frac{1}{\sqrt{n}} \sum_{i=1}^n |e_i\rangle$ as one would ordinarily do in a finite-dimensional Hilbert space. This latter example shows that, when n is infinite, ${}^*\text{Hilb}$ features some genuinely new quantum states: $\frac{1}{\sqrt{n}} \sum_{i=1}^n |e_i\rangle$ is *finite*, in the sense that it has finite norm, but not *near-standard*, in the sense that it is not infinitesimally close to any vector in the corresponding standard Hilbert space. These extra states are the key to constructing the interacting quantum observables we need.

The trick to constructing quantum clocks in ${}^*\text{Hilb}$ is to think of all time-translation groups as actually discrete and periodic, at least in the non-standard sense. Consider the abelian group ${}^*\mathbb{Z}_\omega$ formed by the non-standard integers modulo some positive non-standard natural $\omega \in {}^*\mathbb{N}$. When ω is finite, these are the usual finite cyclic groups. When ω is infinite, however, these groups are always very large, and contain \mathbb{Z} as a subgroup. Indeed, we can take the following representatives for the elements of ${}^*\mathbb{Z}_\omega$:

$${}^*\mathbb{Z}_\omega := \left(\left\{ - \left\lfloor \frac{\omega-1}{2} \right\rfloor, \dots, + \left\lfloor \frac{\omega}{2} \right\rfloor \right\}, +, 0 \right) \quad (17)$$

If $i, j \in \mathbb{Z}$ then $i + j$ is always finite and no modular reduction ever occurs, so that addition of i and j in ${}^*\mathbb{Z}_\omega$ is the same as addition in \mathbb{Z} . Now let $\omega_{uv}, \omega_{ir} \in {}^*\mathbb{R}$ be non-infinitesimal positive non-standard reals with $\omega_{uv}\omega_{ir} = \omega \in {}^*\mathbb{N}$ and consider the following subset of ${}^*\mathbb{R}$:

$$\frac{1}{\omega_{uv}} {}^*\mathbb{Z}_\omega := \left\{ \frac{n}{\omega_{uv}} \in {}^*\mathbb{R} \mid n \in \left\{ - \left\lfloor \frac{\omega-1}{2} \right\rfloor, \dots, + \left\lfloor \frac{\omega}{2} \right\rfloor \right\} \right\} \quad (18)$$

The subset $\frac{1}{\omega_{uv}} {}^*\mathbb{Z}_\omega$ inherits the group structure of ${}^*\mathbb{Z}_\omega$. The uv/ir suffixes for the numbers ω_{uv} and ω_{ir} originate from a habit, typical of quantum field theory, to distinguish between “infra-red” infinities – arising because space is infinitely large – and “ultra-violet” infinities – arising because space is infinitely fine: the parameter ω_{uv} controls how fine the subdivision of ${}^*\mathbb{R}$ specified by $\frac{1}{\omega_{uv}} {}^*\mathbb{Z}_\omega$ is, while the parameter $\omega_{ir} = \omega/\omega_{uv}$ controls how large a portion of ${}^*\mathbb{R}$ it covers.

For different choices of parameters $\omega_{uv}, \omega_{ir} \in {}^*\mathbb{R}$, the discrete periodic non-standard groups $\frac{1}{\omega_{uv}} {}^*\mathbb{Z}_\omega$ can be used to approximate all the time-translation groups which we are interested in. In what follows, we write $\left(\frac{1}{\omega_{uv}} {}^*\mathbb{Z}_\omega \right)_{\text{fin}}$ for the subgroup formed by the finite elements, i.e. by those elements which are finite reals.² If $x \in \left(\frac{1}{\omega_{uv}} {}^*\mathbb{Z}_\omega \right)_{\text{fin}}$, we write $\text{st}(x)$ for the unique standard real which is infinitesimally close to x .

² Note to the reader versed in non-standard analysis: this is an *external* subgroup and is only used for the purpose of connecting the non-standard groups to their standard counterparts. It is never used in any constructions within ${}^*\text{Hilb}$.

► **Proposition 4.** *Let $\omega_{uv}, \omega_{ir} \in {}^*\mathbb{R}$ be non-infinitesimal positive non-standard reals such that $\omega := \omega_{uv}\omega_{ir} \in {}^*\mathbb{N}$ is integer. The time-translation groups G for discrete periodic, discrete, continuous periodic and continuous dynamics are exactly the standard groups which can be obtained as quotient by infinitesimals of the subgroup of finite elements of $\frac{1}{\omega_{uv}}{}^*\mathbb{Z}_\omega$:*

$$G = \text{st} \left(\left(\frac{1}{\omega_{uv}}{}^*\mathbb{Z}_\omega \right)_{\text{fin}} \right) \quad (19)$$

More specifically, we have the following combinations:

1. if ω_{uv} is finite and ω_{ir} is finite we obtain the discrete periodic case $G = \frac{1}{\text{st}(\omega_{uv})}\mathbb{Z}_\omega \cong \mathbb{Z}_\omega$;
2. if ω_{uv} is finite and ω_{ir} infinite we obtain the discrete case $\frac{1}{\text{st}(\omega_{uv})}\mathbb{Z} \cong \mathbb{Z}$;
3. if ω_{uv} is infinite and ω_{ir} is finite we obtain the continuous periodic case $\mathbb{R}/\text{st}(\omega_{ir})\mathbb{Z}$;
4. if ω_{uv} is infinite and ω_{ir} is infinite we obtain the continuous case \mathbb{R} .

Hence all standard time-translation groups listed above can be approximated, up to infinitesimals, by the subgroup of finite elements of a discrete periodic non-standard group.

3.3 Quantum clocks from interacting quantum observables – Take II

Armed with our dagger compact category ${}^*\text{Hilb}$ and with approximations of our favourite time-translation groups by discrete periodic non-standard groups, we are finally in a position to define our quantum clocks.

► **Definition 5.** *A quantum clock is a pair of interacting quantum observables $(\mathcal{G}, \odot, \bullet)$ in ${}^*\text{Hilb}$ with \odot special commutative, equipped with a group isomorphism $(\mathbb{K}(\odot), \blacktriangle, \blacklozenge) \cong \frac{1}{\omega_{uv}}{}^*\mathbb{Z}_\omega$ for some non-infinitesimal positive $\omega_{uv}, \omega_{ir} \in {}^*\mathbb{R}$ with $\omega := \omega_{uv}\omega_{ir} \in {}^*\mathbb{N}$. We refer to $\frac{1}{\omega_{uv}}{}^*\mathbb{Z}_\omega$ as the time-translation group and to $\text{st} \left(\left(\frac{1}{\omega_{uv}}{}^*\mathbb{Z}_\omega \right)_{\text{fin}} \right)$ as the associated standard time-translation group. We refer to the classical states $\mathbb{K}(\odot)$ of \odot as the (clock) time states – which we index as $|t\rangle$ using the elements $t \in \frac{1}{\omega_{uv}}{}^*\mathbb{Z}_\omega$ – and to the observable \odot as the clock time observable.*

Note that the specific choice of $\frac{1}{\omega_{uv}}{}^*\mathbb{Z}_\omega$ – i.e. the specific choice of parameters ω_{uv}, ω_{ir} – is part of the data of a quantum clock. We will only mention it explicitly when relevant, to lighten the notation.

► **Proposition 6.** *Quantum clocks with non-standard time-translation group $\frac{1}{\omega_{uv}}{}^*\mathbb{Z}_\omega$ exist for all non-infinitesimal positive $\omega_{uv}, \omega_{ir} \in {}^*\mathbb{R}$ with $\omega := \omega_{uv}\omega_{ir} \in {}^*\mathbb{N}$.*

In the case of infinite-dimensional quantum clocks, working in the non-standard setting gives us access to a lot of states and linear maps which would not be well-defined in the standard setting, let alone continuous. On a quantum clock with time-translation group $\frac{1}{\omega_{uv}}{}^*\mathbb{Z}_\omega$, for example, we can construct the following *plane-wave states* indexed by all $E \in \frac{1}{\omega_{ir}}{}^*\mathbb{Z}_\omega$ (note the switch from ω_{uv} to ω_{ir}):

$$|E\rangle := \sum_{t \in \frac{1}{\omega_{uv}}{}^*\mathbb{Z}_\omega} e^{i2\pi \frac{Et}{\omega}} |t\rangle \quad (20)$$

In particular, we have that $\langle E|t\rangle = e^{-i2\pi Et}$. This is exactly the phase that an energy eigenstate with energy E acquires after time t has passed. As the following result shows, this is no coincidence: in a quantum clock $(\mathcal{G}, \odot, \bullet)$, the classical states $|E\rangle$ for \bullet always label the possible energy values that the corresponding dynamical systems can have.

In terms of those endomorphisms, the defining equations for algebras take the following form:

Diagrammatic equations (23):

- Two boxes labeled α_s and α_t stacked vertically are equal to a single box labeled α_{t+s} .
- A box labeled α_0 is equal to a vertical line.
- A box labeled α_t^\dagger is equal to a box labeled α_{-t} .

But these are exactly the equations defining a unitary representation $(\alpha_t)_t$ of the time-translation group! Clearly, we are off to a good start.

Algebras for the monad $_ \otimes \bullet$ form a category, with morphisms $\Phi : \alpha \rightarrow \beta$ from an algebra $\alpha : \mathcal{H} \otimes \mathcal{G} \rightarrow \mathcal{H}$ to another algebra $\beta : \mathcal{K} \otimes \mathcal{G} \rightarrow \mathcal{K}$ given by the linear maps $\Phi : \mathcal{H} \rightarrow \mathcal{K}$ which satisfy the following equation:

Diagrammatic equation (24):

- A box labeled Φ above a box labeled α is equal to a box labeled β above a box labeled Φ .

Because the monad $_ \otimes \bullet$ is obtained from a monoid which is part of a pair $(\circlearrowleft, \bullet)$ of interacting quantum observables with \circlearrowleft commutative, the category of algebras has a symmetric monoidal structure, with the tensor product $\alpha \otimes \beta$ of two algebras defined as follows:

Diagrammatic equation (25):

- Two boxes labeled α and β are connected at their bottom inputs by a green circle with two wires passing through it.

The following result shows that the category of algebras captures exactly quantum dynamical systems, equivariant maps between them and their natural notion of composition.

► **Proposition 9.** *Let $(\mathcal{G}, \circlearrowleft, \bullet)$ be a quantum clock. The algebras α for the dagger Frobenius monad $_ \otimes \bullet$ such that α_t is near-standard for all t correspond to quantum dynamical systems for the standard time-translation, i.e. strongly continuous unitary representations $(\text{st}(\alpha_t))_{\text{st}(t) \in G}$ of the standard time-translation group. Morphisms between algebras correspond to equivariant maps for the representations. Tensor product of algebras corresponds to synchronised composition of quantum dynamical systems:*

Diagrammatic equation (26):

- The expression $(\alpha \otimes \beta)_t$ is defined as a diagram where the tensor product of α and β (from equation 25) is followed by a box labeled t connected to the bottom outputs of the green circle.
- This is equal to $\alpha_t \otimes \beta_t$.

19:12 A Diagrammatic Approach to Quantum Dynamics

► **Definition 10.** Let $(\mathcal{G}, \circlearrowleft, \circlearrowright)$ be a quantum clock. A quantum dynamical system for the quantum clock is an algebra for the dagger Frobenius monad $_ \otimes \circlearrowright$. Morphisms of algebras will be referred to as equivariant maps between quantum dynamical systems. Tensor product of algebras will be referred to as synchronised parallel composition of quantum dynamical systems.

4.2 States and histories

States of a system are a static concept. In a quantum dynamical system, we are instead more interested in the evolution of states under the dynamics:



For a generic dynamical system H – e.g. seen as a topological space – the evolution of a state under the dynamics is usually written as a *flow-line* $\Psi : \mathbb{R} \rightarrow H$, a map from the time object to the dynamical system associating a state $\Psi(t) \in H$ to each instant point t in time (with the obvious generalisation from continuous dynamics to the other three kinds). This is not, however, an exact correspondence in general: a given map $\mathbb{R} \rightarrow H$ is often not going to be the flow-line of a state.

As we mentioned before, the traditional perspective on time in quantum dynamics is that time is an external classical parameter, so the definition of state evolution through flow-lines suffers from the issue described above. In our framework, on the other hand, “time” lives inside the same category as the quantum systems it governs, incarnated into the quantum clocks that tick it. We can exploit the additional algebraic structure available to show that flow-lines, realised inside the category of algebras, correspond exactly to the evolutions of states in quantum dynamical systems.

► **Proposition 11.** Let $(\mathcal{G}, \circlearrowleft, \circlearrowright)$ be a quantum clock and $\alpha : \mathcal{H} \otimes \mathcal{G} \rightarrow \mathcal{H}$ be a quantum dynamical system for it. Then \circlearrowright is also a quantum dynamical system for it – the quantum clock itself, governed by its own time. The morphisms of algebras $\Psi : \circlearrowright \rightarrow \alpha$:



are exactly the evolutions of states of \mathcal{H} under the dynamics of α . We refer to such morphisms as the histories of states.

4.3 Hamiltonians

Hamiltonians are often the very first concept that students of quantum dynamics are introduced to, so it may be surprising that we have not mentioned them so far. The reason for such a delay is that this work adopts a view of dynamics as time-translation symmetry,

rather than as solution of certain differential equations: our objects of primary concern are unitary representations, not their infinitesimal generators. That said the *Hamiltonian* – as the energy observable of a quantum dynamical system – is of paramount physical interest, so we now proceed to characterise it in our framework.

In previous sections, we have considered the dagger Frobenius monad $_ \otimes \bullet$ (with associated dagger Frobenius comonad $_ \otimes \bullet$) and we have seen that the algebras of $_ \otimes \bullet$ capture dynamics. We have also seen, when talking about quantum clocks, that the other quantum observable in the interacting pair, namely \circ , is somehow associated with energy: one naturally wonders whether there is an algebraic connection between \circ – which is dual to \bullet – and Hamiltonians – which are dual to dynamics.

An alternative characterisation of a quantum observable is in terms of complete families of projectors. A complete family of projectors $(P_E : \mathcal{H} \rightarrow \mathcal{H})_{E \in X}$ is characterised by the following equations:

$$\begin{array}{c} \boxed{P_F} \\ \boxed{P_E} \end{array} = \delta_{E,F} \boxed{P_E} \quad \sum_E \boxed{P_E} = \text{---} \quad \boxed{P_E^\dagger} = \boxed{P_E}$$

(29)

If the labels for the projectors P_E are taken from the clock energy states E , as necessary for the projectors associated with a Hamiltonian, then the equations above can be equivalently rewritten diagrammatically as the equations for coalgebras of the dagger Frobenius comonad $_ \otimes \circ$:

$$\begin{array}{c} \text{---} \\ \boxed{\alpha^\dagger} \end{array} = \begin{array}{c} \boxed{\alpha^\dagger} \\ \text{---} \end{array} \quad \begin{array}{c} \text{---} \\ \boxed{\alpha^\dagger} \\ \text{---} \end{array} = \text{---} \quad \begin{array}{c} \text{---} \\ \boxed{\alpha^\dagger} \\ \text{---} \end{array} = \begin{array}{c} \boxed{\alpha} \\ \text{---} \end{array}$$

(30)

In the literature, these are also referred to as *projector-valued spectra* [8]. We can give such coalgebras an operational interpretation as coherent versions of quantum measurements: if we feed a state $|\psi\rangle$ of \mathcal{H} in input, we obtain in output an entangled state $\sum_E P_E |\psi\rangle \otimes |E\rangle$ of $\mathcal{H} \otimes \mathcal{G}$. Subsequently measuring \mathcal{G} in the $(|E\rangle)_E$ basis yields the usual von Neumann non-demolition measurement corresponding to the complete family of orthogonal projectors $(P_E)_E$: if outcome E is observed, the state in \mathcal{H} has collapsed to $P_E |\psi\rangle$.

Given a quantum dynamical system α , we now show how to obtain the coalgebra for $_ \otimes \circ$ corresponding to its Hamiltonian. We will do so by proving Schrödinger's Equation. In its differential version, Schrödinger's Equation states that if $|\psi_E\rangle$ is an energy eigenstate with energy E then the evolution of $|\psi_E\rangle$ in a quantum dynamical system α is given by the following equation:

$$i\hbar \frac{d}{dt} \alpha_t |\psi_E\rangle = E |\psi_E\rangle \tag{31}$$

The following exponentiated version of Schrödinger's Equation provides the symmetry equivalent of the usual differential equation:

$$\alpha_t |\psi_E\rangle = e^{-i2\pi Et} |\psi_E\rangle \tag{32}$$

where we have chosen energy and time units such that $\hbar := 2\pi\hbar = 1$.

5.1 Weyl Canonical Commutation Relations

Traditionally, the Heisenberg Canonical Commutation Relations characterise the duality between position and momentum observables in the differential generators picture. The Weyl Canonical Commutation Relations characterise the corresponding duality of position and momentum observables in the symmetry picture, by specifying a braiding relation between the space-translation symmetry and the momentum-boost symmetry. When time is quantised, the duality between time-translation symmetry T_t and energy-shift symmetry S_E in a quantum clock can similarly be characterised by the Weyl CCRs, as follows:

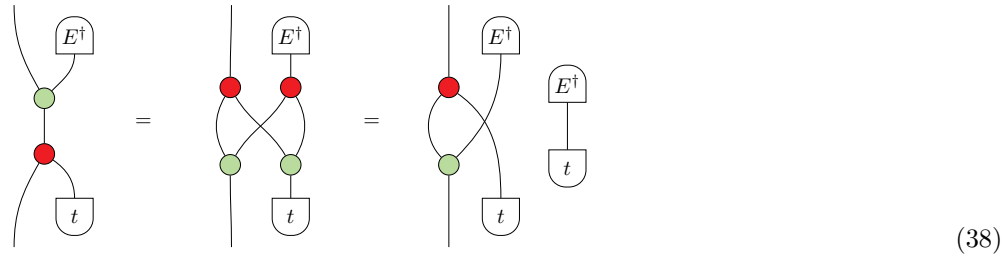
$$S_E T_t = e^{i2\pi Et} T_t S_E \tag{36}$$

This can be equivalently expressed in terms of the adjoint of S_E , to match the exact formulation of our result below:

$$S_E^\dagger T_t = e^{-i2\pi Et} T_t S_E^\dagger \tag{37}$$

In our formalism, the Weyl CCRs are an immediate consequence of a diagrammatic axiom of interacting quantum observables.

► **Proposition 14.** *The Weyl CCRs for time and energy duality are an immediate consequence of the bialgebra law:*



recalling that $\langle E|t\rangle = e^{-i2\pi Et}$.

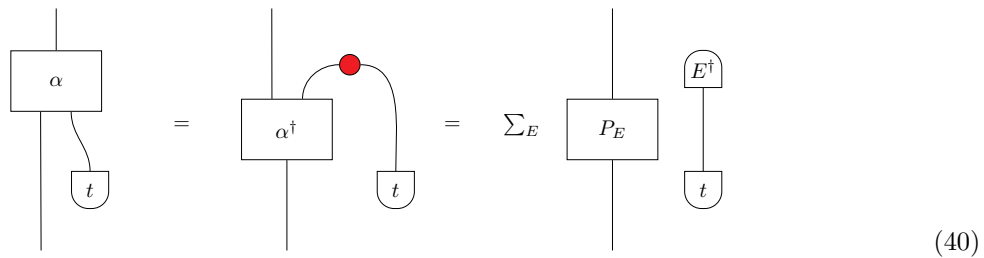
5.2 Stone's Theorem on 1-parameter unitary groups

Stone's Theorem on 1-parameter unitary groups is a key result in dynamics, showing that dynamics can be uniquely reconstructed from the Hamiltonian observable. In the symmetry picture, it can be stated as follows:

$$\alpha_t = \int_{G^\wedge} e^{-i2\pi Et} P_E dE \tag{39}$$

where G is the time-translation group, α is the quantum dynamical system and P_E are the projectors on the energy eigenspaces. We are working in the non-standard settings, so that the integral $\int_{G^\wedge} dE$ is really just a sum. [25]

► **Proposition 15.** *Stone's Theorem is a consequence of diagrammatic time-energy duality:*



recalling that $\langle E|t\rangle = e^{-i2\pi Et}$.

5.3 von Neumann’s Mean Ergodic Theorem

In a rather precise sense, von Neumann’s Mean Ergodic Theorem is the inverse of Stone’s Theorem, showing that the Hamiltonian observable can be reconstructed from the dynamics. The usual formulation of von Neumann’s Theorem only talks about the ground energy eigenspace, but an equivalent formulation can be used to reconstruct all energy eigenspaces:

$$P_E = \frac{1}{|G|} \int_G e^{i2\pi Et} \alpha_t dt f \tag{41}$$

where G is the time-translation group, α is the quantum dynamical system and P_E are the projectors on the energy eigenspaces. By $|G|$ we literally mean the size of G , which is a well-defined scalar $|G| = \omega$ in the non-standard world: physically, this is the volume $\omega = \omega_{uv}\omega_{ir}$ of time-energy configuration space. The equation above is the limit-free, non-standard equivalent of the usual formulation of von Neumann’s theorem in terms of limits. As with Stone’s Theorem before, the integral $\int_G dt$ is really just a sum \sum in the non-standard setting.

► **Proposition 16.** *von Neumann’s Mean Ergodic Theorem is a consequence of diagrammatic time-energy duality:*

$$\text{Diagrammatic equation (42)} \tag{42}$$

recalling that $\langle t|E \rangle = e^{i2\pi Et}$.

6 Conclusions

In this work, we have presented a diagrammatic framework to reason about quantum dynamics, using algebras and coalgebras for a monad and a comonad induced by a pair of interacting quantum observables. We have been able to treat dynamics, both discrete and continuous, of finite- and infinite-dimensional quantum systems, thanks to the rich tool-set provided by hyperfinite non-standard Hilbert spaces. We have shown that our framework yields completely straightforward diagrammatic proofs of some key results in quantum dynamics.

In future work, we will explore the foundational and computational implications of our new framework. Specifically, we will detail the applications to the problem of time observable and to the formulation of Feynman’s Clock, already sketched in the author’s DPhil Thesis [13].

References

- 1 Samson Abramsky and Bob Coecke. A categorical semantics of quantum protocols. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science, 2004.*, pages 415–425. IEEE, 2004. doi:10.1109/LICS.2004.1319636.
- 2 Samson Abramsky and Bob Coecke. Categorical Quantum Mechanics. In K. Engesser, Gabbay D. M., and Lehmann D., editors, *Handbook of Quantum Logic and Quantum Structures*, pages 261–323. Elsevier, 2009. doi:10.1016/B978-0-444-52869-8.50010-4.

- 3 Samson Abramsky and Chris Heunen. H^* -algebras and nonunital Frobenius algebras: first steps in infinite-dimensional categorical quantum mechanics. *Clifford Lectures, AMS Proceedings of Symposia in Applied Mathematics*, 71:1–24, 2012. URL: <http://arxiv.org/abs/1011.6123>.
- 4 Jeremy Butterfield. On time in quantum physics. In Heather Dyke and Adrian Bardon, editors, *A Companion to the Philosophy of Time*. John Wiley & Sons Ltd, 2014.
- 5 Bob Coecke and Ross Duncan. Interacting quantum observables: categorical algebra and diagrammatics. *New Journal of Physics*, 13(4):043016, 2011. doi:10.1088/1367-2630/13/4/043016.
- 6 Bob Coecke, Ross Duncan, Aleks Kissinger, and Quanlong Wang. Strong complementarity and non-locality in categorical quantum mechanics. In *Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer Science*, pages 245–254. IEEE Computer Society, 2012. doi:10.1109/LICS.2012.35.
- 7 Bob Coecke and Aleks Kissinger. *Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning*. Cambridge University Press, 2017. doi:10.1017/9781316219317.
- 8 Bob Coecke and Dusko Pavlovic. Quantum measurements without sums. In G. Chen, L. Kauffman, and S. Lomonaco, editors, *Mathematics of Quantum Computing and Technology*. Taylor and Francis, 2007. URL: <https://arxiv.org/abs/quant-ph/0608035>.
- 9 Bob Coecke, Dusko Pavlovic, and Jamie Vicary. A new description of orthogonal bases. *Mathematical Structures in Computer Science*, 23(3):555–567, 2013. doi:10.1017/S0960129512000047.
- 10 Ross Duncan and Kevin Dunne. Interacting Frobenius Algebras are Hopf. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 535–544. ACM, 2016. doi:10.1145/2933575.2934550.
- 11 Richard P Feynman. Simulating physics with computers. *International journal of theoretical physics*, 21(6):467–488, 1982. doi:10.1007/BF02650179.
- 12 Richard P Feynman. Quantum mechanical computers. *Foundations of physics*, 16(6):507–531, 1986. doi:10.1007/BF01886518.
- 13 Stefano Gogioso. *Categorical Quantum Dynamics*. PhD thesis, University of Oxford, 2017. URL: <https://arxiv.org/abs/1709.09772>.
- 14 Stefano Gogioso and Fabrizio Genovese. Infinite-dimensional Categorical Quantum Mechanics. *Electronic Proceedings in Theoretical Computer Science*, 236:51–69, 2017. doi:10.4204/EPTCS.236.4.
- 15 Stefano Gogioso and Fabrizio Genovese. Towards Quantum Field Theory in Categorical Quantum Mechanics. *Electronic Proceedings in Theoretical Computer Science*, 266:349–366, 2018. doi:10.4204/EPTCS.266.22.
- 16 Stefano Gogioso and William Zeng. Generalised Mermin-type non-locality arguments. *Logical Methods in Computer Science*, 15(2), 2019. URL: <https://lmcs.episciences.org/5402>.
- 17 Robert Goldblatt. *Lectures on the hyperreals. An introduction to nonstandard analysis*. Springer-Verlag, 1998.
- 18 Chris Heunen and Martti Karvonen. Monads on dagger categories. *Theory and Applications of Categories*, 31(35):1016–1043, 2016. URL: <https://arxiv.org/abs/1602.04324>.
- 19 Jan Hilgevoord. Time in quantum mechanics: a story of confusion. *Studies In History and Philosophy of Science Part B: Studies In History and Philosophy of Modern Physics*, 36(1):29–60, 2005. doi:10.1016/j.shpsb.2004.10.002.
- 20 André Joyal and Ross Street. The geometry of tensor calculus, I. *Advances in mathematics*, 88(1):55–112, 1991. doi:10.1016/0001-8708(91)90003-P.
- 21 Aleks Kissinger. *Pictures of Processes: Automated Graph Rewriting for Monoidal Categories and Applications to Quantum Computing*. PhD thesis, University of Oxford, 2012. URL: <https://arxiv.org/abs/1203.0202>.
- 22 Jarrod R. McClean, John A. Parkhill, and Alán Aspuru-Guzik. Feynman’s clock, a new variational principle, and parallel-in-time quantum dynamics. *Proceedings of the National Academy of Sciences*, 110(41):E3901–E3909, 2013. doi:10.1073/pnas.1308069110.

- 23 Thomas Pashby. Time and quantum theory: A history and a prospectus. *Studies in History and Philosophy of Science Part B: Studies in History and Philosophy of Modern Physics*, 52:24–38, 2015. doi:10.1016/j.shpsb.2015.03.002.
- 24 Bryan W Roberts. *Time, symmetry and structure: A study in the foundations of quantum theory*. PhD thesis, University of Pittsburgh, 2012. URL: <http://d-scholarship.pitt.edu/12533>.
- 25 Abraham Robinson. *Non-standard analysis*. Princeton University Press, 1974.
- 26 Peter Selinger. A Survey of Graphical Languages for Monoidal Categories. In Bob Coecke, editor, *New Structures for Physics*, volume 813 of *Lecture Notes in Physics*, pages 289–355. Springer, 2011. doi:10.1007/978-3-642-12821-9_4.
- 27 Jamie Vicary. Categorical formulation of finite-dimensional quantum algebras. *Communications in Mathematical Physics*, 304(3):765–796, 2011. doi:10.1007/s00220-010-1138-0.
- 28 Hermann Weyl. Quantenmechanik und gruppentheorie. *Zeitschrift für Physik*, 46(1-2):1–46, 1927. doi:10.1007/BF02055756.
- 29 Hermann Weyl. *The theory of groups and quantum mechanics*. Dover Publications Inc., New York, 1950.
- 30 Stanisław L. Woronowicz. Compact matrix pseudogroups. *Communications in Mathematical Physics*, 111(4):613–665, 1987. doi:10.1007/BF01219077.
- 31 Stanisław L. Woronowicz. Compact quantum groups. *Symétries quantiques (Les Houches, 1995)*, 845(884):98, 1998.

A

 Categorical Quantum Mechanics

Categorical quantum mechanics takes its roots in the seminal work [1, 2] and a detailed treatment of the first decade of work in the field can be found in the 900+ page monograph [7]. Here we recap some fundamentals of the formalism, for the benefit of readers from different communities who may be unfamiliar with them.

A.1 Symmetric monoidal categories

The general motivation behind the application of category-theoretic tools lies in the intuition that the features distinguishing quantum theory from classical physics can be understood in terms of the way quantum processes compose, sequentially and in parallel: this forms the basis of the *process-theoretic* description of quantum theory. The mathematical arena for such process-theoretic description is that of symmetric monoidal categories:

- physical systems are objects;
- processes between systems are morphisms;
- sequential composition of processes is composition of morphisms;
- parallel composition of processes is tensor product of morphisms;
- the process of doing nothing to a system is the identity morphism;
- the tensor product of objects is interpreted as a joint system;
- the tensor unit is interpreted as a trivial system.

Of special interest in categorical quantum mechanics is the symmetric monoidal category \mathbf{fHilb} of finite-dimensional Hilbert spaces and complex linear maps between them (with the tensor product of Hilbert spaces as tensor product of objects and the Kronecker product of complex matrices as tensor product of morphisms).

Monoidal categories have a natural diagrammatic formalism – see [26] for a comprehensive survey – in which systems/objects A are depicted as wires and processes/morphisms $f : A \rightarrow$

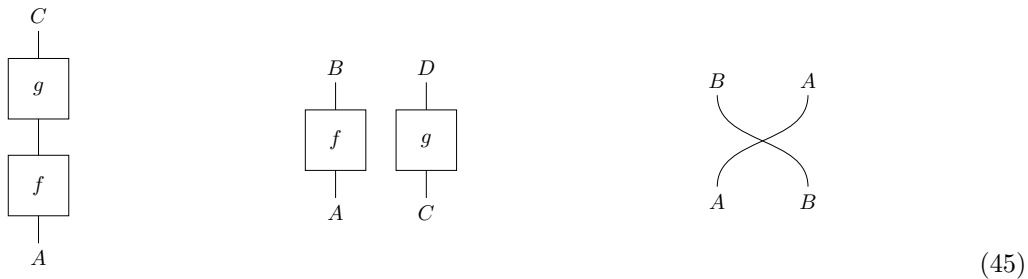
B are depicted as boxes.



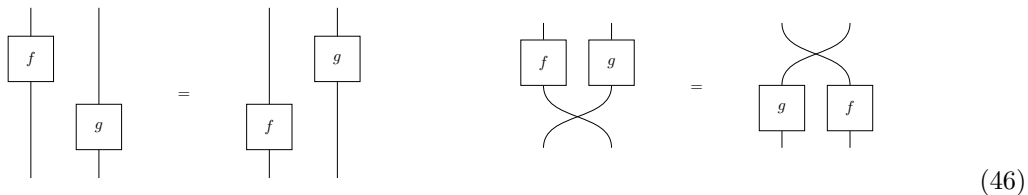
Tensor product of systems is depicted by stacking the corresponding wires side-by-side horizontally. As a convention, we draw our diagrams bottom-to-top, so that the wire(s) corresponding to the domain of a morphism (the *inputs* of the process) are at the bottom and the wire(s) corresponding to the codomain of a morphism (the *outputs* of the process) are at the top, so that a generic morphism/process $f : A_1 \otimes \dots \otimes A_n \rightarrow B_1 \otimes \dots \otimes B_m$ is depicted as follows:



Sequential composition $g \circ f : A \rightarrow C$ of two processes $f : A \rightarrow B$ and $g : B \rightarrow C$ is depicted by stacking the corresponding boxes vertically and connecting the output wires of f to the input wires of g ; parallel composition $f \otimes g : A \otimes C \rightarrow B \otimes D$ of two processes $f : A \rightarrow B$ and $g : C \rightarrow D$ is depicted by stacking the corresponding boxes side-by-side horizontally; the symmetry isomorphisms $A \otimes B \rightarrow B \otimes A$ are depicted by a crossing of the wires:



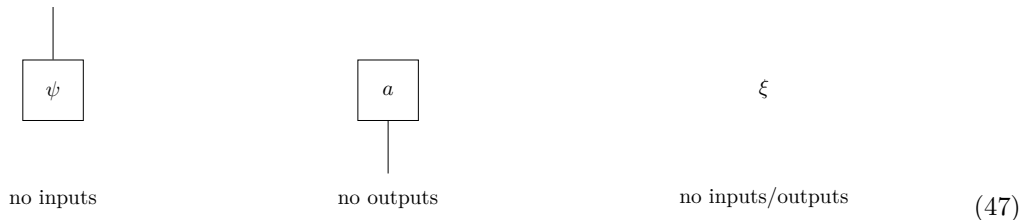
Two diagrams are considered equal if they are equal up to isotopy, keeping the (relative ordering of the) input and output endpoints fixed: this principle is often referred to as “only topology matters”. For example, a special case of bifunctionality for the tensor product holds by “sliding” the two boxes vertically (on the left), while naturality of the symmetry isomorphism is obtained by “sliding boxes through each other” over a wire crossing (on the right):



Planar isotopy is sufficient for monoidal categories. For symmetric monoidal categories, on the other hand, a little amount of 4d space is used for sliding across wire crossings.

A.2 States, scalars and effects

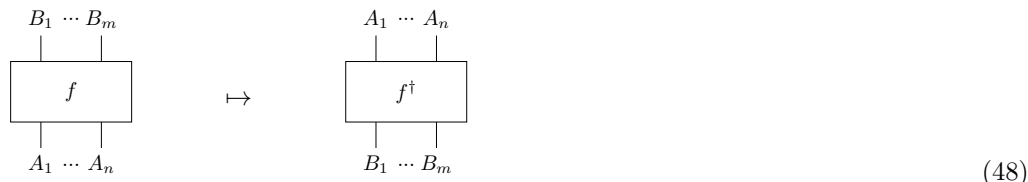
Special cases of boxes are those without any input and/or any output wires:



Boxes with no input wires are called *states* and they correspond to the process of producing something in a system starting from nothing (aka the trivial system). In the category \mathbf{fHilb} , states correspond exactly to vectors in a Hilbert space, i.e. to kets $|\psi\rangle$. Boxes with no inputs nor outputs are called *scalars*. In the category \mathbf{fHilb} , scalars correspond to complex numbers ξ and we will write them as floating numbers. Finally, boxes with no outputs are called *effects*. In the category \mathbf{fHilb} , effects correspond exactly to covectors in a Hilbert space, i.e. to bras $\langle a|$. Effects can be thought of as the process of conditioning on the outcome of a (non-degenerate demolition) quantum measurement: applied to a state in a system, an effect returns a complex number (the norm squared of which yields the outcome probability, according to the Born rule).

A.3 Dagger symmetric monoidal categories

The symmetric monoidal categories of interest in categorical quantum mechanics are equipped with an involutive op-functor, the *dagger*, which sends morphisms $f : A \rightarrow B$ to morphisms $f^\dagger : B \rightarrow A$. In the diagrammatic formalism, the dagger is depicted as a vertical mirror symmetry:



In the dagger symmetric monoidal category \mathbf{fHilb} , the dagger is (chosen to be) the operation of taking the adjoint (i.e. the conjugate transpose of matrices). In particular, the dagger sends a state $|\psi\rangle$ to the corresponding effect $\langle\psi|$ and vice-versa. On scalars, the dagger acts as complex conjugation $\xi^\dagger = \xi^*$.

A.4 Quantum observables

Some boxes have special significance and a special notation is reserved to them. The most important case is that of *symmetric special \dagger -Frobenius algebras* $\odot := (\mathcal{H}, \dot{\circlearrowleft}, \dot{\circlearrowright}, \circlearrowleft, \circlearrowright)$: these are depicted by coloured dots with wires connected to them – lovingly referred to as *spiders* in the literature – and the axioms defining them are given in the main text.

The reason why special symmetric \dagger -Frobenius algebras are of key interest in categorical quantum mechanics is their correspondence in the category \mathbf{fHilb} to quantum observables, i.e. to finite-dimensional C^* -algebras [27]. In particular, *commutative* special \dagger -Frobenius algebras – often referred to as *classical structures* in the literature – correspond bijectively to

orthonormal bases, where the basis vectors $|g\rangle$ in each basis are given by the *classical states* for the algebra, i.e. those satisfying the Equation (5). For a given orthonormal basis $(|g\rangle)_g$, the comultiplication is obtained as $\varphi := \sum_g (|g\rangle \otimes |g\rangle) \langle g|$, i.e. the map $|g\rangle \mapsto |g\rangle \otimes |g\rangle$, the counit as $\phi := \sum_g \langle g|$, i.e. the map $|g\rangle \mapsto 1$, the multiplication μ and unit η as their adjoints. More generally, the orthogonal projectors $p : \mathcal{H} \rightarrow \mathcal{H}$ in a quantum observable can be characterised as the central, self-adjoint, idempotent elements for the algebra.³

A.5 Dagger compact structure

In the category \mathbf{fHilb} , each system \mathcal{H} is equipped with a classical structure \odot for each choice of orthonormal basis. Each such classical structure – and more generally each symmetric special \dagger -Frobenius algebras on \mathcal{H} – induces a self-duality on \mathcal{H} through its cup and cap, because of the snake equation (3), which holds by planar isotopy. The cup and cap are symmetric and related by the dagger, making \mathbf{fHilb} a *dagger compact category*.

A.6 Infinite-dimensional categorical quantum mechanics

The main obstacle to the extension of categorical quantum mechanics to infinite dimensions is the disappearance of symmetric special \dagger -Frobenius algebras: while it is true that for a complete orthonormal basis one can still define the comultiplication $\varphi = \sum_{n=1}^{\infty} (|n\rangle \otimes |n\rangle) \langle n|$ and multiplication μ [3], the unit/counit would give rise to infinite-norm states $\phi = \sum_{n=1}^{\infty} |n\rangle$. While [3] suggests it may be possible to overcome the absence of units in this context, the non-existence of infinite group algebras – necessary to this work – is intrinsically related to the presence of infinities and cannot be fixed directly.

Enter non-standard analysis. Because *approximate* units $\phi^{(\nu)} = \sum_{n=1}^{\nu} |n\rangle$ for the algebra associated to a basis exist for all $\nu \in \mathbb{N}$, by Transfer Theorem [17, 25] we can take ν to be some *infinite* natural number and obtain a genuine unit $\phi = \sum_{n=1}^{\nu} |n\rangle$, multiplication, counit and comultiplication $\varphi = \sum_{n=1}^{\nu} (|n\rangle \otimes |n\rangle) \langle n|$ for a special commutative \dagger -Frobenius algebra. Moreover, by Transfer Theorem we can also formulate group algebras for all abelian groups with ν elements. The full details can be found in [14, 15] and in Section 3.5 of the author's DPhil thesis [13].

TL;DR: we work in the dagger compact category ${}^*\mathbf{Hilb}$ of non-standard Hilbert spaces with dimension some non-standard natural number $\nu \in {}^*\mathbb{N}$: from the non-standard perspective these spaces are finite-dimensional, so the Transfer Theorem can be used to lift many of the structures and properties of the dagger compact category \mathbf{fHilb} . In particular, the algebraic manipulation of vectors, matrices and scalars in ${}^*\mathbf{Hilb}$ is analogous to that of their \mathbf{fHilb} counterparts. When related back to standard Hilbert spaces, however, the objects of ${}^*\mathbf{Hilb}$ cover much more than \mathbf{fHilb} , including both the separable infinite-dimensional Hilbert spaces used in traditional quantum mechanics and the non-separable ones arising in quantum field theory.

B Proofs

Proof of Proposition 3

Proof. Substitute the antipode for its definition and apply the snake equation for \odot . ◀

³ See [27] for the full proof and Section 2.4.2 of the author's DPhil thesis [13] for a summary, noting that a left-to-right diagrammatic convention is adopted in the latter.

Proof of Proposition 4

Proof. If we take ω_{uv} finite and ω_{ir} finite, then $\omega = N \in \mathbb{N}$, all elements in $\frac{1}{\omega_{uv}} \star \mathbb{Z}_\omega$ are finite and taking the standard part yields $\frac{1}{\text{st}(\omega_{uv})} \mathbb{Z}_\omega$. If we take ω_{uv} finite and ω_{ir} infinite, then the finite elements in $\frac{1}{\omega_{uv}} \star \mathbb{Z}_\omega$ are those in the form $\frac{1}{\omega_{uv}} \mathbb{Z}$ and taking the standard part yields $\frac{1}{\text{st}(\omega_{uv})} \mathbb{Z}$. If we take ω_{uv} infinite and ω_{ir} finite, then we have the following subgroup inclusion

$$\frac{1}{\omega_{uv}} \star \mathbb{Z}_\omega = \frac{\omega_{ir}}{\omega} \star \mathbb{Z}_\omega = \left\{ \frac{n\omega_{ir}}{\omega} \mid n \in \star \mathbb{Z}_\omega \right\} \subset \star \mathbb{R} / \omega_{ir} \star \mathbb{Z} \quad (49)$$

All elements are finite and taking the standard part yields $\mathbb{R} / \text{st}(\omega_{ir}) \mathbb{Z}$. If we take ω_{ir} infinite and ω_{uv} infinite, finally, we have the following subset inclusion:

$$\frac{1}{\omega_{uv}} \star \mathbb{Z}_\omega = \left\{ \frac{n}{\omega_{uv}} \mid n \in \left\{ - \left\lfloor \frac{\omega-1}{2} \right\rfloor, \dots, + \left\lfloor \frac{\omega}{2} \right\rfloor \right\} \right\} \subset \star \mathbb{R} \quad (50)$$

The finite elements cover the finite elements of $\star \mathbb{R}$ with infinitesimal mesh, hence taking the standard part yields \mathbb{R} . ◀

Proof of Proposition 6

Proof. Because $\omega := \omega_{uv}\omega_{ir} \in \star \mathbb{N}$, by the Transfer Principle we always have an object \mathcal{G} of $\star \text{Hilb}$ with orthonormal basis $(|e_i\rangle)_{i \in \star \mathbb{Z}_\omega}$. Let \odot be the special commutative \dagger -Frobenius algebra associated to the orthonormal basis, define \bullet to be $|0\rangle$ and \blacklozenge to be the linear extension of the multiplication in $\frac{1}{\omega_{uv}} \star \mathbb{Z}_\omega$. Then $(\mathcal{G}, \odot, \bullet)$ is a pair of interacting quantum observables in $\star \text{Hilb}$ corresponding to the group algebra $\star \mathbb{C}[\frac{1}{\omega_{uv}} \star \mathbb{Z}_\omega]$, as we wanted. ◀

Proof of Proposition 7

Proof. The possible values of energy E must correspond bijectively with the possible unitary group homomorphisms $G \rightarrow \mathbb{C}$ yielding the phases acquired under time-translation by energy eigenstates. Canonically, such homomorphisms are the elements of the Pontryagin dual G^\wedge .

Checking that the plane-waves $|E\rangle$ are the classical states for \bullet is straightforward. Because clock time states form an orthonormal basis, we can biject the effects $\langle E|$ with the multiplicative characters $\chi_E : t \mapsto e^{-i2\pi Et} \in G^\wedge$. The (adjoint of the) copy condition is multiplicativity of characters $\chi_E(t+s) = \chi_E(t)\chi_E(s)$. The (adjoint of the) delete condition is the condition that $\chi_E(0) = 1$. The (adjoint of the) self-conjugacy condition, finally, is unitarity of characters $\chi_E(t)^\dagger = \chi_E(-t)$.

Under the identification of $\langle E|$ with χ_E , it is immediate to see that \blacklozenge acts as pointwise multiplication of characters and that \odot corresponds to the trivial character, so that G^\wedge is obtained by taking the standard part of the finite elements in $\frac{1}{\omega_{ir}} \star \mathbb{Z}_\omega$. ◀

Proof of Proposition 9

Proof. By requiring α_t to be near-standard for all t we have singled out exactly those representations of the non-standard group which yield representations $\text{st}(\alpha_{\text{st}(t)})$ for the corresponding standard group. The defining equations for unitary representations are already satisfied. The defining equation for a morphism $\Phi : \alpha \rightarrow \beta$ implies that $\Phi\alpha_t = \beta_t\Phi$ for all t , so that morphism of algebras give equivariant maps of representations. Synchronised parallel composition speaks for itself. ◀

Proof of Proposition 11

Proof. The proof is entirely by straightforward diagrammatic manipulation, based on the observation that Ψ is exactly the time evolution of the state $\Psi(0)$:

$$\Psi = \Psi \circledast \Psi = \alpha \circledast \Psi \quad (51)$$

◀

Proof of Proposition 12

Proof. That α^\dagger is a coalgebra for $_ \otimes \bullet$ is a straightforward diagrammatic check using the algebra equations for α and the snake equations. The fact that Schrödinger's Equation holds for the eigenstates of projectors is another straightforward diagrammatic check. ◀

Proof of Propositions 14, 15 and 16

The proofs are already essentially in the respective diagrammatic statements.

CARTOGRAPHER: A Tool for String Diagrammatic Reasoning

Paweł Sobociński

University of Southampton, UK

Paul W. Wilson

University of Southampton, UK and University College, London, UK

Fabio Zanasi

University College London, UK

Abstract

We introduce CARTOGRAPHER, a tool for editing and rewriting string diagrams of symmetric monoidal categories. Our approach is principled: the layout exploits the isomorphism between string diagrams and certain cospans of hypergraphs; the implementation of rewriting is based on the soundness and completeness of convex double-pushout rewriting for string diagram rewriting.

2012 ACM Subject Classification Software and its engineering → Visual languages

Keywords and phrases tool, string diagram, symmetric monoidal category, graphical reasoning

Digital Object Identifier 10.4230/LIPIcs.CALCO.2019.20

Category Tool Paper

Funding *Fabio Zanasi*: acknowledges support from EPSRC grant EP/R020604/1.

Acknowledgements The authors thank Aleks Kissinger for valuable discussion, and the anonymous referees for their comments and feedback.

1 Introduction

String diagrammatic theories are increasingly important in computer science. They have been recently been used in a number of applications, including enabling the simplification of quantum circuits using the ZX-calculus [10], compositional descriptions of models of concurrency such as Petri Nets [18, 6], compositional accounts of signal flow graphs in control theory [7, 11, 1] and Bayesian reasoning [8, 14, 13]. These examples, as well as many others, work with the language of *symmetric monoidal categories* (SMCs). This paper addresses the need for tool support for *symmetric monoidal theories* - graphical rewriting systems of SMCs.

CARTOGRAPHER is a graphical editor and proof assistant for symmetric monoidal theories. It provides a graphical string diagram editor to construct morphisms, and a prover in which rewrite rules can be specified and executed. Further, CARTOGRAPHER has a firm theoretical foundation, its rewriting backend based on recent work in the area [5, 3, 20, 4]. The goal of this paper is to motivate CARTOGRAPHER, explain the basic features of the backend and the front end, and describe some of the technical challenges that were solved in creating it. The tool and its user guide are available on the CARTOGRAPHER website at <http://cartographer.id/>.

Our motivating example is the rewriting system in Figure 1. The intended semantic interpretation is that of binary circuits, where each wire carries an n bit number for some fixed n . Green nodes with two outputs copy numbers, those with no outputs discard their input, while red nodes perform addition modulo 2^n .



© Paweł Sobociński, Paul W. Wilson, and Fabio Zanasi;
licensed under Creative Commons License CC-BY

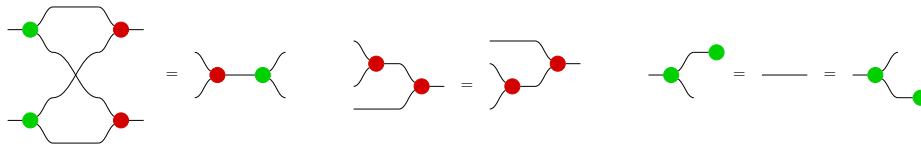
8th Conference on Algebra and Coalgebra in Computer Science (CALCO 2019).

Editors: Markus Roggenbach and Ana Sokolova; Article No. 20; pp. 20:1–20:7

Leibniz International Proceedings in Informatics

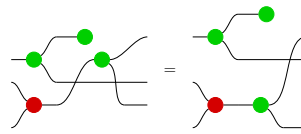


LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** Example rules for binary circuits with copying ($\text{---} \begin{matrix} \bullet \\ \swarrow \searrow \\ \bullet \end{matrix} \text{---}$), adding ($\text{---} \begin{matrix} \bullet \\ \swarrow \swarrow \\ \bullet \end{matrix} \text{---}$), and discarding ($\text{---} \begin{matrix} \bullet \\ \swarrow \swarrow \\ \bullet \end{matrix} \text{---}$).

As well as the rules in Figure 1, this rewriting system implicitly uses three *generators*; atomic sub-diagrams, each with some number of inputs and outputs. These are the *copy* ($\text{---} \begin{matrix} \bullet \\ \swarrow \searrow \\ \bullet \end{matrix} \text{---}$), *add* ($\text{---} \begin{matrix} \bullet \\ \swarrow \swarrow \\ \bullet \end{matrix} \text{---}$), and *discard* ($\text{---} \begin{matrix} \bullet \\ \swarrow \swarrow \\ \bullet \end{matrix} \text{---}$) operations. The laws of symmetric monoidal categories permit moving generators around up to an isotopy made precise in [15, 19]. For our purposes, it suffices to say informally that generators can be slid along wires, and moved around on the page, but not rotated. By way of example, consider the equivalent diagrams in Figure 2.



■ **Figure 2** Example of string diagrams considered equal under the laws of SMCs.

CARTOGRAPHER allows reasoning modulo the laws of symmetric monoidal categories. The user can deform morphisms up to the SMC laws without making proofs unsound, and the prover does not require (e.g. when matching the l.h.s. of a rule) the user to explicitly use the laws of symmetric monoidal categories. Put another way, the user should not have to “untangle” the wires of the diagram before applying a rule of some theory.

To put this into context, compare CARTOGRAPHER to two “competing” tools: Quantomatic [17] and Globular [2] (or its more recent descendant, *homotopy.io*). In a sense, CARTOGRAPHER sits between them: providing a more general setting than Quantomatic, while at the same time being more focussed than Globular.

| software | generality | geometric intuition |
|--------------|--------------------|--|
| Quantomatic | compact-closed | generators can implicitly be moved and wires bent back |
| Cartographer | symmetric monoidal | generators can implicitly be moved |
| Globular | higher categories | no implicit deformations permitted |

Quantomatic deals with *compact closed* categories, in which not only may generators be moved, but wires may be “bent backwards”. In terms of our circuit analogy, this would allow for feedback, e.g. as used in a simple latch. CARTOGRAPHER allows such feedback, but as an explicit compact closed structure in the theory at hand, not implicitly assumed to exist by the underlying tool. On the other hand, Globular is much more general, aiming to support diagrammatic reasoning in higher categories. While this allows more freedom, when working with SMCs it comes at the cost of having to explicitly use SMC laws in proofs, e.g. using the functoriality of the monoidal product to slide two generators past each other.

Contributions

The contribution of CARTOGRAPHER is twofold. First, in the back end we implement an algorithm for matching and rewriting modulo the laws of SMC based on the adequacy result of [5]. The algorithm works with a data structure for *Open Hypergraphs*, which we introduce

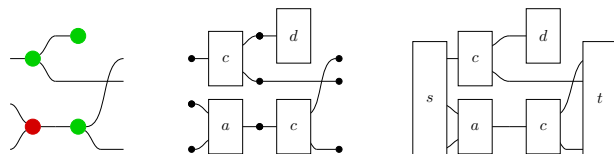
in this paper. Second, in the front end, we use an algorithm for the layout of these directed acyclic open hypergraphs which behaves well under rewriting and deformation of diagrams.

2 Directed Acyclic Open Hypergraphs

The main problem of implementing rewriting modulo symmetric monoidal structure is in finding a data structure in which equivalent terms have a single representation. For example, the two equivalent diagrams of Figure 2 should have the same underlying representation. Our approach is principled, because it uses the isomorphism between equivalent terms and cospans of hypergraphs found in [5]. Starting from this result, we propose an alternative but equivalent representation which is more convenient to work with.

We begin with an overview of the open hypergraphs of [5], and the CARTOGRAPHER data structure – illustrated in Figure 3 along with the corresponding string diagram. Beginning with the central (open) hypergraph, hyperedges are denoted $\overline{i \square j}$, and represent the generators of the string diagram. There are two kinds of nodes, both denoted \bullet . Firstly (ordered) boundary nodes, which are connected to a single wire (input or output, but not both). Secondly, internal nodes, having exactly one input and one output wire. These two conditions are the “monogamicity” requirement of [5], and effectively ensure that the hypergraph corresponds to a string diagram. For full details, see [5, Definition 3.6].

In contrast, the hypergraphs of CARTOGRAPHER are closed, and so nodes are rendered simply as wires, each with exactly one input and one output connection. Boundary nodes are replaced by adding special generators to the signature of the hypergraph, s (boundary source) and t (boundary target). Nodes are then uniquely identified by the two “ports” they connect – a *port* being a specific position on the boundary of a hyperedge.



■ **Figure 3** From left to right: a string diagram, its open hypergraph representation with signature $\Sigma = \{a, c, d\}$, and the equivalent closed hypergraph with signature $\Sigma' = \Sigma \cup \{s, t\}$.

- **Definition 1.** A $k \rightarrow m$ CARTOGRAPHER *hypergraph* (Σ, E, W) consists of:
- the *signature* Σ , which can be thought of as the set of *types* of hyperedges. Each has arity $ar : \Sigma \rightarrow \mathbb{N} \times \mathbb{N}$, giving the number of inputs and outputs. We require that the Σ contains *boundary* generators σ, τ , with $ar(\sigma) = (0, k)$ and $ar(\tau) = (m, 0)$;
 - the set of *hyperedges* E , with a function $typ : E \rightarrow \Sigma$ that assigns types to hyperedges. Moreover, there are *boundary* hyperedges $\{s, t\} \subseteq E$ s.t. $typ^{-1}(\sigma) = \{s\}$, $typ^{-1}(\tau) = \{t\}$;
 - the set of *wires* W . Given a hyperedge $e \in E$, if $ar(typ(e)) = (p, q)$ then we say e has p input ports, denoted e^1, e^2, \dots, e^p , and q output ports denoted e_1, e_2, \dots, e_q . A *wire* $w \in W$ is an ordered pair (e_i, f^j) of a *source port* e_i and a *target port* f^j , denoting a directed connection from the i^{th} output of e to the j^{th} input of f .

3 Visualising and Editing Open Hypergraphs

In contrast to Quantomatic [17] which uses a force-directed layout, and Globular [2] which has a fixed style for morphism layout, we use a *layered graph drawing* algorithm similar to that of Dot [12]. Our reasons for choosing layered graph drawing are as follows. Firstly, it

was an aesthetic choice to represent string diagrams similarly to how they appear in the literature. Secondly, string diagrams drawn with the layered discipline retain a closer link with the underlying algebraic description of morphisms, since the term can be easily read off the string diagram in the form of a composition-of-monoidal-products. Thirdly, in contrast to *force-directed* approaches, the elements of a layered graph layout do not move around on the page, which is problematic from a user-experience perspective, because they are harder for the user to click. Additionally, force-directed layouts can change significantly after a rewrite rule is applied, with little control over the resulting diagram. This can be confusing for the user, because the string diagram may look very different. Finally, using layered hypergraphs offers a simple and intuitive way to enforce acyclicity: users may only connect generators if the target appears to the right of the source.

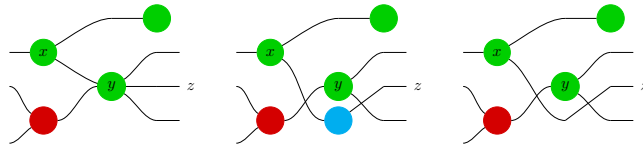
Interactive Layered Graph Drawing

We briefly summarise the interactive layered graph drawing approach of CARTOGRAPHER. By “interactive”, we mean to distinguish CARTOGRAPHER’s layout algorithm from other layered graph drawing approaches – such as Dot’s – in which a static graph is given as input, and positions of nodes and edges are returned. CARTOGRAPHER allows for the *incremental construction* of hypergraphs, meaning that users begin with a blank canvas, and add generators and connections one-by-one. We call it a layered graph drawing approach because it uses two key ideas from those approaches: the user of *layers*, and of *pseudonodes*.

► **Definition 2.** Given a CARTOGRAPHER hypergraph (Σ, E, W) and $e \neq e' \in E$, there is a directed path from e to e' if there exists a sequence (e_1, \dots, e_n) where $e_i \in E$, $e_1 = e$, $e_n = e'$ and for each e_i, e_{i+1} there exist j_1, j_2 such that $((e_i)_{j_1}, (e_{i+1})^{j_2}) \in W$. A *layering* is a function $L : E \rightarrow \mathbb{N}$ such that:

- (i) if there is a directed path from e to e' then $L(e) < L(e')$;
- (ii) for every non-boundary hyperedge $e \in E$, $L(s) < L(e) < L(t)$.

The layering L essentially serves as the “ x coordinate” of each hyperedge. The second idea from layered graph layout is the use of *pseudonodes*, which are conceptually related to the edge-points of Dixon and Kissinger’s Open Graphs [9], but used here only for layout purposes: they prevent wires from crossing generators. For a concrete example of why this is desirable, consider Figure 4. In the left-hand diagram, the wire from x to z passes through y and it is not clear whether x is connected to y and y to z , or if x is directly connected to z . Inserting pseudonodes into the graph clears up the ambiguity.



■ **Figure 4** Left, a diagram with only generators (rendered \bullet and \bullet), center, the same diagram after inserting pseudonodes (rendered \bullet), and right, the diagram as it appears with pseudonodes hidden.

The Layout Algorithm

We briefly outline the layout algorithm used in CARTOGRAPHER. Because the algorithm is interactive, it takes the form of a *layout state*, and a number of *actions* that the user can take. We model these actions as functions of the layout state.

The layout state is a tuple (H, G) of a hypergraph H as in definition 1, and an *integer grid* G , which keeps track of the positions of generators and pseudonodes as two dimensional vectors. Users can perform two actions on the layout state:

1. Placing a generator at a specific position on an integer grid
2. Moving a generator from one position to another
3. Connecting a source port to a target port

Moving and placing a generator is straightforward: if a generator e is moved or placed such that it would overlap with another generator f , then f is moved down within the same layer to make space. However, when connecting ports we must ensure that the hypergraph H remains acyclic. This is enforced using the following constraints:

- If generators e, f have layers such that $L(e) \leq L(f)$, then outputs of f may not be connected to inputs of e .
- If a generator f is reachable from e , then f may not be moved such that $L(f) \leq L(e)$.

These constraints ensure that layering respects the properties of Definition 2, preserving acyclicity. Finally, for every operation, the set of required pseudonodes is maintained, along with their positions in G . In particular, this means updates for any operation which changes connectivity, or modifies the number of layers between two generators.

4 Matching, Convexity, Rewriting

As well as an interactive string diagram editor, CARTOGRAPHER enables diagrammatic reasoning. A derivation consists of a series of rewrites, using a set of rules specified by the user. A *rule* consists of two CARTOGRAPHER hypergraphs, the lhs and the rhs, with identical boundaries. Rewriting is implemented by double-pushout rewriting of hypergraphs, with soundness and completeness guaranteed by [5, Theorem 5.6].

Applying a rule to a string diagram consists of three steps: finding a match for the lhs a rule, checking for convexity, and applying the rewrite rule. A *match* is an hypergraph embedding (an injective, homomorphic mapping of hyperedges and nodes) of open hypergraphs, with one subtlety: the boundary ports of the pattern match can map to non-boundary ports in the target. CARTOGRAPHER builds matches incrementally by using the backtracking logic library *logict* [16]. Roughly speaking, wires and generators are added to the working match until either there are no more unmatched wires or generators, or a contradiction is reached, in which case the search backtracks. Candidate matches are then checked for *convexity* [5], which is needed for a rewrite to be valid modulo the laws of SMCs. Roughly speaking, all directed paths that start and end in a matched region must remain within the match. Once a convex match has been identified, the internal hyperedges of the matched region are removed and replaced with the right hand side of the rewrite rule.

The CARTOGRAPHER UI shows a list of matches of rules found in the current proof term. Users can apply rewrites by hovering over each match to see which part of the graph will be rewritten, and then clicking to apply the rewrite.

5 Conclusions and Future Work

CARTOGRAPHER is still in early stages of development. We are working on:

- improving the layout algorithms by adapting heuristics from other tools that work with layered graphs;

- more advanced features for diagrammatic reasoning, including support for structured proofs (using e.g. user-generated Lemmas) and adapting other user-friendly features originally developed for theorem provers and proof assistants;
- higher level specification features, such as support for bang-boxes, recursive definitions, and proof strategies;
- better decoupling between the rewriting back end and the layout front end, enabling extensions such as rewriting modulo compact closed structure.
- support for rewriting without the convexity condition, which would allow rewriting modulo a chosen Frobenius structure [5, 20]. This would be useful as symmetric monoidal categories with a chosen Frobenius structure (also called hypergraph categories) are a special kind of compact-closed categories, and find applications in the study of quantum processes, dynamical systems and natural language processing, among other areas.

References

- 1 John Baez and Jason Erbele. Categories In Control. *Theory and Applications of Categories*, 30:836–881, 2015. URL: <http://arxiv.org/abs/1405.6881>.
- 2 Krzysztof Bar, Aleks Kissinger, and Jamie Vicary. Globular: an online proof assistant for higher-dimensional rewriting. In *Leibniz International Proceedings in Informatics*, volume 52, pages 34:1–34:11, 2016. ncatlab.org/nlab/show/Globular.
- 3 Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Paweł Sobociński, and Fabio Zanasi. Confluence of Graph Rewriting with Interfaces. In *Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, pages 141–169, 2017. doi:10.1007/978-3-662-54434-1_6.
- 4 Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Paweł Sobociński, and Fabio Zanasi. Rewriting with Frobenius. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 165–174, 2018. doi:10.1145/3209108.3209137.
- 5 Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Paweł Sobociński, and Fabio Zanasi. Rewriting modulo symmetric monoidal structure. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science - LICS '16*, pages 710–719, New York, NY, USA, 2016. ACM Press. doi:10.1145/2933575.2935316.
- 6 Filippo Bonchi, Joshua Holland, Robin Piedeleu, Paweł Sobociński, and Fabio Zanasi. Diagrammatic algebra: from linear to concurrent systems. *PACMPL*, 3(POPL):25:1–25:28, 2019. URL: <https://dl.acm.org/citation.cfm?id=3290338>, doi:10.1145/3290338.
- 7 Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. The Calculus of Signal Flow Diagrams I: Linear relations on streams. *Inf. Comput.*, 252:2–29, 2017.
- 8 Benjamin Cabrera, Tobias Heindel, Reiko Heckel, and Barbara König. Updating Probabilistic Knowledge on Condition/Event Nets using Bayesian Networks. In *29th International Conference on Concurrency Theory, CONCUR 2018, September 4-7, 2018, Beijing, China*, pages 27:1–27:17, 2018. doi:10.4230/LIPIcs.CONCUR.2018.27.
- 9 Lucas Dixon and Aleks Kissinger. Open-graphs and monoidal theories. *Mathematical Structures in Computer Science*, 23(2):308–359, 2013.
- 10 Andrew Fagan and Ross Duncan. Optimising Clifford Circuits with Quantomatic. *Electronic Proceedings in Theoretical Computer Science*, 287:85–105, January 2019. doi:10.4204/EPTCS.287.5.
- 11 Brendan Fong, Paolo Rapisarda, and Paweł Sobociński. A categorical approach to open and interconnected dynamical systems. In *Thirty-first annual ACM/IEEE symposium on Logic and Computer Science (LiCS 2016)*, pages 495–504, 2016. doi:10.1145/2933575.2934556.

- 12 E.R. Gansner, E. Koutsofios, S.C. North, and K.-P. Vo. A technique for drawing directed graphs. *IEEE Transactions on Software Engineering*, 19(3):214–230, March 1993. doi: 10.1109/32.221135.
- 13 Bart Jacobs, Aleks Kissinger, and Fabio Zanasi. Causal Inference by String Diagram Surgery. *CoRR*, 2019. URL: <http://arxiv.org/abs/1811.08338>.
- 14 Bart Jacobs and Fabio Zanasi. The Logical Essentials of Bayesian Reasoning. In Joost-Peter Katoen Gilles Barthe and Alexandra Silva, editors, *Probabilistic Programming*. Cambridge University Press, Cambridge, 2019. URL: <http://arxiv.org/abs/1804.01193>.
- 15 Andre Joyal and Ross Street. The Geometry of Tensor Calculus, I. *Adv. Math.*, 88:55–112, 1991.
- 16 Oleg Kiselyov, Chung-chieh Shan, Daniel P Friedman, and Amr Sabry. Backtracking, Interleaving, and Terminating Monad Transformers. *SIGPLAN Not.*, page 12, 2005.
- 17 Aleks Kissinger and Vladimir Zamdzhev. Quantomatic: A Proof Assistant for Diagrammatic Reasoning. *arXiv:1503.01034 [cs, math]*, 9195:326–336, 2015. arXiv: 1503.01034. doi: 10.1007/978-3-319-21401-6_22.
- 18 José Meseguer and Ugo Montanari. Petri nets are monoids. *Information and Computation*, 88(2):105–155, October 1990. doi:10.1016/0890-5401(90)90013-8.
- 19 Peter Selinger. A survey of graphical languages for monoidal categories. *arXiv:0908.3347 [math]*, 813:289–355, 2010. arXiv: 0908.3347. doi:10.1007/978-3-642-12821-9_4.
- 20 Fabio Zanasi. Rewriting in Free Hypergraph Categories. In *Proceedings Third Workshop on Graphs as Models, GaM@ETAPS 2017, Uppsala, Sweden, 23rd April 2017.*, pages 16–30, 2017. doi:10.4204/EPTCS.263.2.

