Reiko Heckel
Stefan Milius (Eds.)

LNCS 8089

Algebra and Coalgebra in Computer Science

5th International Conference, CALCO 2013
Warsaw, Poland, September 2013
Proceedings

Springer

Lecture Notes in Computer Science 8089

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

Reiko Heckel Stefan Milius (Eds.)

Algebra and Coalgebra in Computer Science

5th International Conference, CALCO 2013
Warsaw, Poland, September 3-6, 2013
Proceedings

Springer

Volume Editors

Reiko Heckel
University of Leicester, Department of Computer Science
University Road
Leicester, LE1 7RH, UK
E-mail: reiko@mcs.le.ac.uk

Stefan Milius
Friedrich-Alexander Universität Erlangen-Nürnberg
Lehrstuhl für Theoretische Informatik
Martenstr. 3
91058 Erlangen-Nürnberg, Germany
E-mail: stefan.milius@fau.de

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-40205-0 e-ISBN 978-3-642-40206-7
DOI 10.1007/978-3-642-40206-7
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2013944581

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

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

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

Printed on acid-free paper

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

Preface

CALCO, the International Conference on Algebra and Coalgebra in Computer Science, is a high-level, bi-annual event formed by joining CMCS (the International Workshop on Coalgebraic Methods in Computer Science) and WADT (the Workshop on Algebraic Development Techniques). CALCO aims to bring together researchers and practitioners with interests in foundational aspects, and both traditional and emerging uses of algebras and coalgebras in computer science. The study of algebra and coalgebra relates to the data, process, and structural aspects of software systems.

Previous CALCO editions took place in Swansea (UK, 2005), Bergen (Norway, 2007), Udine (Italy, 2009) and Winchester (UK, 2011). CALCO 2013, the fifth conference in the series, took place in Warsaw (Poland), September 3–6, 2013.

CALCO 2013 received 33 submissions, out of which 18 were selected for presentation at the conference. The standard of submissions was generally very high. The selection process was carried out by the Program Committee, taking into account the originality, quality, and relevance of the material presented in each submission, based on the opinions of three or four expert reviewers for each submission. The selected and revised papers are included in this volume, together with contributions by the invited speakers Andrej Bauer, Mikołaj Bojańczyk, Neil Ghani, and Damien Pous.

CALCO 2013 was co-located with two workshops. The CALCO Early Ideas Workshop, CALCO EI, was dedicated to presentation of work in progress and original research proposals. PhD students and young researchers were particularly encouraged to contribute. CALCO EI was organized by Monika Seisenberger. The CALCO Tools Workshop, organized by Lutz Schröder, is dedicated to tools based on algebraic and/or coalgebraic principles. These tool papers also appear in this volume.

We wish to thank all the authors for submitting their papers to CALCO 2013, the Program Committee for its diligent work in the selection process, and the external reviewers for their support in evaluating papers.

We are grateful to the University of Warsaw and the Polish Mathematical Society for hosting CALCO 2013 and to the Organizing Committee, chaired by Bartek Klin and Andrzej Tarlecki, for all the local arrangements. We also thank the Warsaw Centre of Mathematics and Computer Science for their financial support. At Springer, Alfred Hofmann and his team supported the publishing process. We gratefully acknowledge the use of EasyChair, the conference management system by Andrei Voronkov.

June 2013

Reiko Heckel
Stefan Milius

Organization

CALCO Steering Committee

Jiří Adámek	Technical University of Braunschweig, Germany
Michel Bidoit	CNRS and ENS de Cachan, France
Corina Cîrstea	University of Southampton, UK
Andrea Corradini	University of Pisa, Italy
José Luiz Fiadeiro	Royal Holloway, University of London, UK
H. Peter Gumm (Co-chair)	Philipps University Marburg, Germany
Rolf Hennicker	Ludwig-Maximilians-University Munich, Germany
Bart Jacobs	Radboud University Nijmegen, Germany
Bartek Klin	University of Warsaw, Italy
Hans-Jörg Kreowski	University of Bremen, Germany
Alexander Kurz	University of Leicester, UK
Marina Lenisa	University of Udine, Italy
Ugo Montanari	University of Pisa, Italy
Larry Moss	Indiana University, Bloomington, USA
Till Mossakowski (Co-chair)	DFKI Lab Bremen and University of Bremen, Germany
Fernando Orejas	Politechnical University Catalunia, Barcelona, Spain
Francesco Parisi-Presicce	University of Rome La Sapienza, Italy
Dirk Pattinson	Australian National University, Australia
John Power	University of Bath, UK
Horst Reichel	Technical University of Dresden, Germany
Jan Rutten	CWI Amsterdam and Radboud University Nijmegen, The Netherlands
Lutz Schröder	Friedrich-Alexander University Erlangen-Nürnberg, Germany
Andrzej Tarlecki	University of Warsaw, Poland

Program Committee

Luca Aceto	Reykjavik University, Iceland
Jiří Adámek	Technical University of Braunschweig, Germany
Lars Birkedal	IT University of Copenhagen, Denmark
Filippo Bonchi	CNRS, ENS-Lyon, France

Corina Cîrstea	University of Southhampton, UK
Bob Coecke	University of Oxford, UK
Andrea Corradini	University of Pisa, Italy
Mai Gehrke	Université Paris Diderot – Paris 7, France
H. Peter Gumm	Philipps University Marburg, Germany
Gopal Gupta	University of Texas at Dallas, USA
Ichiro Hasuo	Tokyo University, Japan
Reiko Heckel (Co-chair)	University of Leicester, UK
Bart Jacobs	Radboud University Nijmegen, The Netherlands
Ekaterina Komendantskaya	University of Dundee, Scotland, UK
Barbara König	University of Duisburg-Essen, Germany
José Meseguer	University of Illinois, Urbana-Champaign, USA
Marino Miculan	University of Udine, Italy
Stefan Milius (Co-chair)	Friedrich-Alexander University Erlangen-Nürnberg, Germany
Larry Moss	Indiana University, Bloomington, USA
Till Mossakowski	DFKI Lab Bremen and University of Bremen, Germany
Prakash Panangaden	McGill University, Montreal, Canada
Dirk Pattinson	Australian National University, Australia
Dusko Pavlovic	Royal Holloway, University of London, UK
Daniela Petrişan	University of Leicester, UK
John Power	University of Bath, UK
Jan Rutten	CWI Amsterdam and Radboud University Nijmegen, The Netherlands
Lutz Schröder	Friedrich-Alexander University Erlangen-Nürnberg, Germany
Monika Seisenberger	Swansea University, UK
Alexandra Silva	Radboud University Nijmegen and CWI Amsterdam, The Netherlands
Paweł Sobociński	University of Southhampton, UK
Sam Staton	University of Cambridge, UK
Yde Venema	University of Amsterdam, The Netherlands
Uwe Wolter	University of Bergen, Norway

Additional Reviewers

Giorgio Bacci	Mihai Codescu	Giuseppe Greco
Ulrich Berger	Josée Desharnais	Helle Hvid Hansen
Arnar Birgisson	Brian Devries	Chris Heunen
Aleš Bizjak	Ross Duncan	Tom Hirschowitz
Roberto Bruni	Murdoch Gabbay	Jean-Baptiste Jeannin
Martin Churchill	Rajeev Goré	Henning Kerstan

Aleks Kissinger
Robbert Krebbers
Sebastian Küpper
Clemens Kupke
Alberto Lluch Lafuente
Guy McCusker
Robert Myers
Marco Peressotti
Vaughan Pratt
Joshua Sack
Mehrnoosh Sadrzadeh
Isar Stubbe
Viktor Winschel
Joost Winter
Fabio Zanasi

Table of Contents

Invited Talks

An Effect System for Algebraic Effects and Handlers 1
Andrej Bauer and Matija Pretnar

Automata and Algebras for Infinite Words and Trees 17
Mikołaj Bojańczyk

Positive Inductive-Recursive Definitions 19
Neil Ghani, Lorenzo Malatesta, and Fredrik Nordvall Forsberg

Coalgebraic Up-to Techniques 34
Damien Pous

Contributed Papers

Exploiting Algebraic Laws to Improve Mechanized Axiomatizations 36
Luca Aceto, Eugen-Ioan Goriac, Anna Ingolfsdottir, Mohammad Reza Mousavi, and Michel A. Reniers

Positive Fragments of Coalgebraic Logics 51
Adriana Balan, Alexander Kurz, and Jiří Velebil

Many-Valued Relation Lifting and Moss' Coalgebraic Logic 66
Marta Bílková and Matěj Dostál

Saturated Semantics for Coalgebraic Logic Programming 80
Filippo Bonchi and Fabio Zanasi

Presenting Distributive Laws 95
Marcello M. Bonsangue, Helle Hvid Hansen, Alexander Kurz, and Jurriaan Rot

Interaction and Observation: Categorical Semantics of Reactive Systems Trough Dialgebras 110
Vincenzo Ciancia

Homomorphisms of Coalgebras from Predicate Liftings 126
Sebastian Enqvist

From Kleisli Categories to Commutative C^*-Algebras: Probabilistic Gelfand Duality 141
Robert Furber and Bart Jacobs

Trace Semantics via Generic Observations 158
Sergey Goncharov

Full Abstraction for Fair Testing in CCS 175
Tom Hirschowitz

A Simple Case of Rationality of Escalation 191
Pierre Lescanne

Coalgebras with Symmetries and Modelling Quantum Systems 205
Daniel Marsden

From Operational Chu Duality to Coalgebraic Quantum Symmetry 220
Yoshihiro Maruyama

Noninterfering Schedulers: When Possibilistic Noninterference Implies Probabilistic Noninterference 236
Andrei Popescu, Johannes Hölzl, and Tobias Nipkow

Simulations and Bisimulations for Coalgebraic Modal Logics 253
Daniel Gorín and Lutz Schröder

A Coalgebraic View of ε-Transitions 267
Alexandra Silva and Bram Westerbaan

Nets, Relations and Linking Diagrams 282
Paweł Sobociński

A Logic-Programming Semantics of Services 299
Ionuţ Ţuţu and José Luiz Fiadeiro

CALCO-Tools Workshop

Preface to CALCO-Tools 314
Lutz Schröder

Checking Conservativity with Hets 315
Mihai Codescu, Till Mossakowski, and Christian Maeder

The HI-Maude Tool 322
Muhammad Fadlisyah and Peter Csaba Ölveczky

Constructor-Based Inductive Theorem Prover 328
Daniel Găină, Min Zhang, Yuki Chiba, and Yasuhito Arimoto

A Timed CTL Model Checker for Real-Time Maude 334
Daniela Lepri, Erika Ábrahám, and Peter Csaba Ölveczky

Hybridisation at Work 340
Renato Neves, Alexandre Madeira, Manuel A. Martins, and Luís S. Barbosa

`Penrose`: Putting Compositionality to Work for Petri Net Reachability 346
Paweł Sobociński and Owen Stephens

QStream: A Suite of Streams 353
Joost Winter

Author Index 359

An Effect System for Algebraic Effects and Handlers

Andrej Bauer and Matija Pretnar

Faculty of Mathematics and Physics
University of Ljubljana

Abstract. We present an effect system for algebraic effects and handlers. Because handlers may transform an effectful computation into a pure one, the effect system is non-monotone in the sense that effects do not just accumulate, but may also be deleted from types or generally transformed. We also provide denotational semantics for the effect system, based on a domain-theoretic model with partial equivalence relations. The semantics validates equational reasoning about effectful computations.

1 Introduction

An *effect system* supplements a traditional type system for a programming language with information about which computational effects may, will, or will not happen when a piece of code is executed. A well designed and solidly implemented effect system helps the programmer understand the code and find mistakes, but it can also be used to safely rearrange, optimize, and parallelize code [1,2]. As many before us [1,3,4,5] we take on the task of striking just the right balance between simplicity and expressivity by devising an effect system for the programming language Eff [6]. The novelty here is that Eff has not only *first-class algebraic effects* [7], but also *effect handlers* [8]. Therefore, an effect system for Eff is by its nature *non-monotone* — an effectful computation may become pure when enclosed by a handler — so effects do not just accumulate in the types, but also get deleted and generally transformed. Another feature of our effect system is that its denotational semantics validates *equational* reasoning, which is traditionally thought to be in the dominion of pure languages, and can be tricky once effects are included [9].

The paper is organized as follows. In §2 we describe the *core Eff* and an effect system for it. In §3 we give a denotational semantics for core Eff, and use it to validate program transformation rules that can be used for equational reasoning about effectful computations.

2 Core Eff

Eff is a ML-style [10,11] programming language. Effects in ML are not visible in the types. For example, inserting a print statement in the middle of code does

R. Heckel and S. Milius (Eds.): CALCO 2013, LNCS 8089, pp. 1–16, 2013.

not create any changes in the typing information. In contrast, in the monadic style such a change would taint all enclosing types with the I/O monad [12]. It is our intention to augment the types with information about computational effects which is unobtrusive for programmers, i.e., in the implementation we envision *effect inference* which serves primarily as a program analysis tool.

In ML-style languages effects are provided through built-in functions and special-purpose constructs such as exceptions and references. Eff is based on the *algebraic* approach in which effects are accessed uniformly and exclusively through *operations*, which are a primitive concept. Examples of operations are reading and writing on a communication channel, updating and looking up the contents of a reference, and raising an exception. Thus, in Eff each terminating computation results either in an (effect-free) value, or it triggers an operation. Each operation has an associated (delimited) *continuation*, which is a suspended computation expecting the result of the operation and doing whatever is to be done after the operation is performed.

Operations by themselves do not actually perform effects. Instead their behavior is controlled by a second primitive notion, the *effect handlers*. These are a direct generalization of exception handlers to other operations. The most significant difference between exception handlers and effect handlers is that the latter have access to the continuation of the handled operation. With handlers we may implement a great variety of computational effects, such as transactional memory, various non-deterministic execution strategies, stream redirection, cooperative multi-threading, delimited continuations, and others.

The current implementation of Eff includes a number of features, such as syntactic sugar, products, records, inductive types, type definitions, effect definitions, etc., which are inessential for a conceptual analysis. We therefore restrict attention to *core Eff* whose syntax is shown below.

Types

$$\begin{aligned}
\text{Effect type } E &::= \texttt{effect}\ (\texttt{operation}\ \mathtt{op}_i : A_i \to B_i)_i\ \texttt{end} \\
\text{Expression type } A, B &::= \texttt{nat} \mid \texttt{bool} \mid \texttt{unit} \mid \texttt{empty} \mid E^{\{\iota_1, \ldots, \iota_n\}} \mid \\
&\qquad A \to \underline{C} \mid \underline{C} \Rightarrow \underline{D} \\
\text{Computation type } \underline{C}, \underline{D} &::= A ! \{\iota_1 \,\#\, \mathtt{op}_1, \ldots, \iota_n \,\#\, \mathtt{op}_n\}
\end{aligned}$$

Terms

$$\begin{aligned}
\text{Expression } e &::= x \mid 0 \mid \texttt{succ}\ e \mid \texttt{true} \mid \texttt{false} \mid \\
&\qquad () \mid \iota \mid \texttt{fun}\ x \mapsto c \mid e \,\#\, \mathtt{op} \mid h \\
\text{Handler } h &::= \texttt{handler}\ \texttt{val}\ x \mapsto c_v \mid (e_i \,\#\, \mathtt{op}_i\ x_i\ k_i \mapsto c_i)_i \\
\text{Computation } c &::= \texttt{val}\ e \mid \texttt{let}\ x = c_1\ \texttt{in}\ c_2 \mid \texttt{let rec}\ f\ x = c_1\ \texttt{in}\ c_2 \mid \\
&\qquad \texttt{iszero}\ e \mid \texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2 \mid \texttt{absurd}\ e \mid e_1\ e_2 \mid \\
&\qquad \texttt{with}\ e\ \texttt{handle}\ c
\end{aligned}$$

We describe informally what the various parts mean, and refer the readers to [6] for a more thorough introduction. Eff uses a fine grain call-by-value evaluation strategy [13], which means that it distinguishes effect-free *expressions* and possibly effectful *computations*. There are corresponding *expression* and *computation types*.

2.1 Expressions

An expression is either a variable x, zero 0, a successor $\texttt{succ}\ e$, a boolean value `true` or `false`, the unit `()`, an effect instance ι, a function abstraction of a computation, an operation $e\,\texttt{\#}\,\texttt{op}$, or a handler transforming computations to computations. We briefly comment on the ones that are peculiar to Eff.

An *effect type*

$$\texttt{effect}\ (\texttt{operation}\ \texttt{op}_i : A_i \to B_i)_i\ \texttt{end}$$

declares operations $\texttt{op}_i$ with given types of parameters A_i and results B_i. (Here and elsewhere, $(\cdots)_i$ indicates that $\cdots$ may be repeated finitely many times.) For example, the effect type of exceptions is

```
effect
    operation abort : unit → empty
end
```

and the effect type `ref` for a reference holding a natural number is

```
effect
    operation lookup : unit → nat
    operation update : nat → unit
end
```
(1)

Effect instances ι are a way of making several copies of the same computational effect. For example, there may be several communication channels, several mutable references, etc. In this respect Eff differs from [5], where bare operations are considered; in terms of Eff that is like having a single instance of each effect.

The expression type $E^{\{\iota_1,\ldots,\iota_n\}}$ is inhabited by expressions which evaluate to one of the instances $\iota_1, \ldots, \iota_n$, whose effect type is E. We call $\{\iota_1, \ldots, \iota_n\}$ a *region* and abbreviate it as ρ. A smaller region is more informative, so ideally we would like all of them to be just singletons. But this is not possible because instances are first-class values and so we can write

$$\texttt{let}\ x = (\texttt{if}\ b\ \texttt{then}\ \iota_1\ \texttt{else}\ \iota_2)\ \texttt{in}\ \cdots$$

The best we can say about x is that its type is $E^{\{\iota_1,\iota_2\}}$.

2.2 Computations

A *computation type* $A!\{\iota_1 \,\#\, \mathtt{op}_1, \ldots, \iota_n \,\#\, \mathtt{op}_n\}$ means that the computation either produces a (pure) value of type A, or triggers one of the listed operations $\iota_1 \,\#\, \mathtt{op}_1, \ldots, \iota_n \,\#\, \mathtt{op}_n$. We abbreviate such finite sets of operations with the letter δ and call them *dirt*. A computation is either a pure value $\mathtt{val}\ e$, a $\mathtt{let}$ binding, a recursive function definition, a zero-test $\mathtt{iszero}\ e$, a conditional statement, a destructor for the $\mathtt{empty}$ type, an application, or a $\mathtt{handle}$ construct.

The computation $\mathtt{val}\ e$ is pure and indicates a "final" result e, while an operation applied to a parameter $\iota \,\#\, \mathtt{op}\, e$ is the principal way of triggering an effect. By itself $\iota \,\#\, \mathtt{op}\, e$ is just a suspended computation with an associated continuation. For an actual effect to take place it has to be handled by a handler, as described below. The continuation associated with $\iota \,\#\, \mathtt{op}\, e$ is $\mathtt{fun}\ x \mapsto \mathtt{val}\ x$. Such operations are known as *generic effects* [7].

A binding $\mathtt{let}\ x = c_1\ \mathtt{in}\ c_2$ is evaluated as follows:

1. if c_1 evaluates to $\mathtt{val}\ e$ then the binding evaluates to c_2 with x bound to e,
2. if c_1 evaluates to an operation $\iota \,\#\, \mathtt{op}\, e$ with continuation κ, then the binding evaluates to $\iota \,\#\, \mathtt{op}\, e$ with continuation $\mathtt{fun}\ y \mapsto (\mathtt{let}\ x = \kappa\, y\ \mathtt{in}\ c_2)$.

It may help to think of $\mathtt{val}$ and $\mathtt{let}$ as being similar to Haskell $\mathtt{return}$ and $\mathtt{do}$, respectively. In ML $\mathtt{val}$ is invisible, while $\mathtt{let}$ is essentially the same as ours.

The handling construct applies a handler to a computation. If h is the *handler*

$$\mathtt{handler}\ \mathtt{val}\ x \mapsto c_v \mid (e_i \,\#\, \mathtt{op}_i\, x_i\, k_i \mapsto c_i)_i$$

and c is a computation then $\mathtt{with}\ h\ \mathtt{handle}\ c$ first evaluates c and then evaluates the clause of the handler that matches the result given by c. If no clause matches, c evaluated to an operation which is propagated outwards. In all cases the handler wraps itself around the continuation so that subsequent operations are handled as well. More precisely:

1. if c evaluates to $\mathtt{val}\ e$, then the handling construct evaluates to c_v with x bound to e,
2. if c evaluates to $e_i \,\#\, \mathtt{op}_i\, e'$ with continuation κ, then the handling construct evaluates to c_i with x_i and k_i bound to e' and $\mathtt{fun}\ y \mapsto \mathtt{with}\ h\ \mathtt{handle}\ \kappa\, y$, respectively.
3. if c evaluates to any other operation $\iota \,\#\, \mathtt{op}\, e'$ with continuation κ, then the handling construct acts as if h contained the clause

$$\iota \,\#\, \mathtt{op}\, x\, k \mapsto (\mathtt{let}\ y = \iota \,\#\, \mathtt{op}\, x\ \mathtt{in}\ k\, y).$$

Thus it evaluates to $\iota \,\#\, \mathtt{op}\, e'$ with continuation $\mathtt{fun}\ y \mapsto \mathtt{with}\ h\ \mathtt{handle}\ (\kappa\, y)$.

We may wrap several handling constructs around c, in which case the inner handler takes precedence. Note that $\mathtt{let}\ x = c_1\ \mathtt{in}\ c_2$ is equivalent to

$$\mathtt{with}\ (\mathtt{handler}\ \mathtt{val}\ x \mapsto c_2)\ \mathtt{handle}\ c_1$$

so we could theoretically omit $\mathtt{let}$.

2.3 Typing Rules

The typing system of core Eff has two typing judgments,

$$\Gamma \vdash_{\mathsf{e}} e : A \qquad \text{and} \qquad \Gamma \vdash_{\mathsf{c}} c : \underline{C}$$

stating that an expression e has expression type A, and that a computation c has computation type $\underline{C}$, respectively. As usual, Γ is a typing context

$$x_1 : A_1, \ldots, x_n : A_n$$

which assigns expression types A_i to distinct variables x_i. We also have subtyping of expression types $A \leq A'$ and computation types $\underline{C} \leq \underline{C'}$. We fix an assignment Ξ of effect types to instances, and assume implicitly that all dirts and regions appearing in the rules are well formed, i.e., if $\iota \mathbin{\#} \mathtt{op}$ appears then in fact $(\iota : E) \in \Xi$ and $\mathtt{op} \in E$, and that a region ρ in E^ρ mentions only instances whose effect type is E.

The typing rules for variables, primitive constants and functions are standard:

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash_{\mathsf{e}} x : A} \qquad \Gamma \vdash_{\mathsf{e}} 0 : \mathtt{nat} \qquad \frac{\Gamma \vdash_{\mathsf{e}} e : \mathtt{nat}}{\Gamma \vdash_{\mathsf{e}} \mathtt{succ}\ e : \mathtt{nat}} \qquad \Gamma \vdash_{\mathsf{e}} () : \mathtt{unit}$$

$$\Gamma \vdash_{\mathsf{e}} \mathtt{true} : \mathtt{bool} \qquad \Gamma \vdash_{\mathsf{e}} \mathtt{false} : \mathtt{bool} \qquad \frac{\Gamma, x : A \vdash_{\mathsf{c}} c : \underline{C}}{\Gamma \vdash_{\mathsf{e}} (\mathtt{fun}\ x \mapsto c) : A \to \underline{C}}$$

The typing rule for instances

$$\frac{\iota \in \rho}{\Gamma \vdash_{\mathsf{e}} \iota : E^\rho}$$

verifies that the instance ι is in the region ρ and consults Ξ to check that ρ contains only instances whose effect type is E. An operation $e \mathbin{\#} \mathtt{op}$ has a function type, where the dirt in the codomain must contain operations $\iota \mathbin{\#} \mathtt{op}$ for ι ranging over the region associated with e:

$$\frac{\Gamma \vdash_{\mathsf{e}} e : E^\rho \qquad (\mathtt{op} : A \to B) \in E \qquad \forall \iota \in \rho \,.\, (\iota \mathbin{\#} \mathtt{op}) \in \delta}{\Gamma \vdash_{\mathsf{e}} e \mathbin{\#} \mathtt{op} : A \to B ! \delta}$$

The handler type $A ! \delta \Rightarrow \underline{C}$ expresses the fact that a handler transforms computations of type $A ! \delta$ to computations of type $\underline{C}$. The typing rule for handlers

$$\frac{\begin{array}{c}\Gamma, x : A \vdash_{\mathsf{c}} c_v : \underline{C} \\ \Gamma \vdash_{\mathsf{e}} e_i : E_i^{\rho_i} \qquad (\mathtt{op}_i : A_i \to B_i) \in E_i \qquad \Gamma, x_i : A_i, k_i : B_i \to \underline{C} \vdash_{\mathsf{c}} c_i : \underline{C} \\ \forall (\iota \mathbin{\#} \mathtt{op}) \in \delta \,.\, \exists i \,.\, \rho_i = \{\iota\} \wedge \mathtt{op}_i = \mathtt{op}\end{array}}{\Gamma \vdash_{\mathsf{e}} (\mathtt{handler}\ \mathtt{val}\ x \mapsto c_v \mid (e_i \mathbin{\#} \mathtt{op}_i\ x_i\ k_i \mapsto c_i)_i) : A ! \delta \Rightarrow \underline{C}}$$

checks that the instances and operations are paired up correctly according to their effect types, all clauses have computation type $\underline{C}$ under suitable assumptions on bound variables, and every operation in δ is handled by a clause. Note

that an operation $\iota \,\#\, \mathtt{op}$ is taken to be handled by a clause $e_i \,\#\, \mathtt{op}_i \, x_i \, k_i \mapsto c_i$ only if $\mathtt{op} = \mathtt{op}_i$ and e_i is ascertained to have exactly the region $\{\iota\}$. If the region were $\{\iota, \iota'\}$ we could not tell whether the clause handles $\iota \,\#\, \mathtt{op}$ or $\iota' \,\#\, \mathtt{op}$.

The typing rules for computations hold no suprises at all:

$$\frac{\Gamma \vdash_{\mathsf{e}} e : A}{\Gamma \vdash_{\mathsf{c}} \mathtt{val}\ e : A!\delta} \qquad \frac{\Gamma \vdash_{\mathsf{c}} c_1 : A!\delta \qquad \Gamma, x : A \vdash_{\mathsf{c}} c_2 : B!\delta}{\Gamma \vdash_{\mathsf{c}} (\mathtt{let}\ x = c_1\ \mathtt{in}\ c_2) : B!\delta}$$

$$\frac{\Gamma, f : A \to \underline{C}, x : A \vdash_{\mathsf{c}} c_1 : \underline{C} \qquad \Gamma, f : A \to \underline{C} \vdash_{\mathsf{c}} c_2 : \underline{D}}{\Gamma \vdash_{\mathsf{c}} (\mathtt{let\ rec}\ f\, x = c_1\ \mathtt{in}\ c_2) : \underline{D}}$$

$$\frac{\Gamma \vdash_{\mathsf{e}} e : \mathtt{nat}}{\Gamma \vdash_{\mathsf{c}} \mathtt{iszero}\ e : \mathtt{bool}!\delta} \qquad \frac{\Gamma \vdash_{\mathsf{e}} e : \mathtt{bool} \qquad \Gamma \vdash_{\mathsf{c}} c_1 : \underline{C} \qquad \Gamma \vdash_{\mathsf{c}} c_2 : \underline{C}}{\Gamma \vdash_{\mathsf{c}} (\mathtt{if}\ e\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2) : \underline{C}}$$

$$\frac{\Gamma \vdash_{\mathsf{e}} e : \mathtt{empty}}{\Gamma \vdash_{\mathsf{c}} (\mathtt{absurd}\ e) : \underline{C}} \qquad \frac{\Gamma \vdash_{\mathsf{e}} e_1 : A \to \underline{C} \qquad \Gamma \vdash_{\mathsf{e}} e_2 : A}{\Gamma \vdash_{\mathsf{c}} e_1\, e_2 : \underline{C}}$$

$$\frac{\Gamma \vdash_{\mathsf{e}} e : \underline{C} \Rightarrow \underline{D} \qquad \Gamma \vdash_{\mathsf{c}} c : \underline{C}}{\Gamma \vdash_{\mathsf{c}} (\mathtt{with}\ e\ \mathtt{handle}\ c) : \underline{D}}$$

The structural subtyping rules are expected as well [14]:

$$\frac{\Gamma \vdash_{\mathsf{e}} e : A \qquad A \le A'}{\Gamma \vdash_{\mathsf{e}} e : A'} \qquad \frac{\Gamma \vdash_{\mathsf{c}} c : \underline{C} \qquad \underline{C} \le \underline{C}'}{\Gamma \vdash_{\mathsf{c}} c : \underline{C}'} \qquad A \le A$$

$$\frac{A \le A' \qquad A' \le A''}{A \le A''} \qquad \frac{A' \le A \qquad \underline{C} \le \underline{C}'}{(A \to \underline{C}) \le (A' \to \underline{C}')} \qquad \frac{\underline{C}' \le \underline{C} \qquad \underline{D} \le \underline{D}'}{(\underline{C} \Rightarrow \underline{D}) \le (\underline{C}' \Rightarrow \underline{D}')}$$

$$\frac{\rho \subseteq \rho'}{E^{\rho} \le E^{\rho'}} \qquad \frac{A \le A' \qquad \delta \subseteq \delta'}{A!\delta \le A'!\delta'}$$

There are two subsumption rules, one for each typing judgment. The subtyping relation is reflexive and transitive. Function and handler types are contravariant in the domain and covariant in the codomain. Indeed, we may always pretend that a handler handles fewer operations, and triggers more operations than it actually does. An effect type is covariant in the region, while a computation type is covariant in both its parts.

Subtyping buys us additional expressivity. For instance, assuming ι_1 and ι_2 are instances of E,

$$\begin{aligned}
&\mathtt{let}\ a = \iota_1\ \mathtt{in}\\
&\mathtt{let}\ b = (\mathtt{if}\ p\ \mathtt{then}\ a\ \mathtt{else}\ \iota_2)\ \mathtt{in}\\
&\qquad \mathtt{val}\ (\mathtt{handler}\ \mathtt{val}\ x \mapsto \mathtt{val}\ ()\ \mid a \,\#\, \mathtt{op}\, x\, k \mapsto \mathtt{val}\ ())
\end{aligned}$$

we would like to give a the type $E^{\{\iota_1\}}$ so that the handler can have the more specific type $A!\{\iota_1 \,\#\, \mathtt{op}\} \Rightarrow \mathtt{unit}!\varnothing$, but then without subtyping $E^{\{\iota_1\}} \leq E^{\{\iota_1,\iota_2\}}$ we could not give b its type $E^{\{\iota_1,\iota_2\}}$.

2.4 Example: State Handlers

We demonstrate the features of our type system by looking at a state handler for the reference effect `ref` shown in (1). Define `state` to be the function

$$\begin{aligned}\mathtt{fun}\ r \mapsto (\mathtt{handler}\ \mathtt{val}\ x \mapsto \mathtt{val}\ (\mathtt{fun}\ s \mapsto \mathtt{val}\ x)\\ \mid r \,\#\, \mathtt{lookup}\ x\ k \mapsto \mathtt{val}\ (\mathtt{fun}\ s \mapsto k\ s\ s)\\ \mid r \,\#\, \mathtt{update}\ s'\ k \mapsto \mathtt{val}\ (\mathtt{fun}\ s \mapsto k\ ()\ s'))\end{aligned}$$

In words, `state` accepts an instance r and returns a handler which handles lookups and updates on r by using the standard functional encoding of the state monad. For any instance ι, expression type A and dirt δ, the function `state` has the type

$$\mathtt{ref}^{\{\iota\}} \to (A!(\{\iota \,\#\, \mathtt{lookup}, \iota \,\#\, \mathtt{update}\} \cup \delta) \Rightarrow (\mathtt{nat} \to A!\delta)!\delta).$$

Thus, if c is a computation of type

$$A!(\{\iota \,\#\, \mathtt{lookup}, \iota \,\#\, \mathtt{update}\} \cup \delta)$$

then

$$\mathtt{let}\ h = \mathtt{state}\ \iota\ \mathtt{in}\ (\mathtt{with}\ h\ \mathtt{handle}\ c) \tag{2}$$

has the type $(\mathtt{nat} \to A!\delta)!\delta$. That is, we obtained a computation which possibly triggers effects δ and returns a function. Upon application of the function to an initial state effects δ may be triggered again, after which a final result of type A is obtained. In particular, if $\delta = \varnothing$ then (2) is a pure computation.

If we weakened the domain of `state` to $\mathtt{ref}^{\{\iota_1,\iota_2\}}$ then we would not be able to deduce that `state` handled anything, so we would only be able to assign to `state` the less useful type

$$\mathtt{ref}^{\{\iota_1,\iota_2\}} \to (A!\delta \Rightarrow (\mathtt{nat} \to A!\delta)!\delta).$$

We may handle several references by invoking several instantiations of `state`. For example, let c be the computation which swaps the contents of two references:

$$\begin{aligned}&\mathtt{let}\ x_1 = \iota_1 \,\#\, \mathtt{lookup}()\ \mathtt{in}\\ &\mathtt{let}\ x_2 = \iota_2 \,\#\, \mathtt{lookup}()\ \mathtt{in}\\ &\mathtt{let}\ u = \iota_1 \,\#\, \mathtt{update}\ x_2\ \mathtt{in}\\ &\mathtt{let}\ v = \iota_2 \,\#\, \mathtt{update}\ x_1\ \mathtt{in}\ ()\end{aligned}$$

By itself, c has the type

$$\mathtt{unit}!\{\iota_1 \,\#\, \mathtt{lookup}, \iota_1 \,\#\, \mathtt{update}, \iota_2 \,\#\, \mathtt{lookup}, \iota_2 \,\#\, \mathtt{update}\},$$

while

$$\texttt{let}\ h_1 = \texttt{state}\ \iota_1\ \texttt{in}\ (\texttt{with}\ h_1\ \texttt{handle}\ c)$$

has type $(\texttt{nat} \to \texttt{unit}!\{\iota_2\,\#\,\texttt{lookup}, \iota_2\,\#\,\texttt{update}\})!\{\iota_2\,\#\,\texttt{lookup}, \iota_2\,\#\,\texttt{update}\}$. If we handle both instances,

$$\begin{aligned}&\texttt{let}\ h_1 = \texttt{state}\ \iota_1\ \texttt{in}\\ &\texttt{let}\ h_2 = \texttt{state}\ \iota_2\ \texttt{in}\\ &\qquad \texttt{with}\ h_2\ \texttt{handle}\ (\texttt{with}\ h_1\ \texttt{handle}\ c),\end{aligned}$$

we get the pure type $\texttt{nat} \to (\texttt{nat} \to \texttt{unit}!\varnothing)!\varnothing$.

3 Denotational Semantics

An outline of a set-theoretic algebraic semantics of computational effects, as developed by [6,15,16,7,8], is shown in the following table.

Programming language	*Algebra*
effect type	algebraic signature
expression type	set
expression	element
computation type	free algebra
pure computation	generator
effectful computation	algebraic operation
handler	homomorphism of algebras

To properly account for recursion and non-termination we adapt it to a *domain-theoretic* semantics of algebraic effects under which expression and computation types are *domains*. We first give Curry-style semantics in which terms are interpreted without being typed, and then, following John Reynolds [17], we provide a Church-style semantics in which types receive meanings as well. It does not matter much what kind of domains we use, they could be ω-cpos, Scott domains, effective Scott domains, or any other kind of domains that model the basic type-theoretic operations (product, function space, coalesced sum).

3.1 Semantics of Expressions and Computations

Expressions could be modeled with *predomains*, because they are inert pieces of data, free from computational effects, including non-termination. However, the bottom element is useful for denotation of ill-typed expressions and runtime errors. In any case, the predomain nature of expressions will be captured later on by the partial equivalence relations. The domains for computation types are free in a suitable sense, i.e., they enjoy a recursion principle which acts as a substitute for the universality of free algebras. We assume a given set of all instances $\mathbb{I}$ and

a set of all operation symbols $\mathbb{O}$, write $+$ for a disjoint sum, and $\oplus$ for coalesced sum.

To interpret expressions and computations we need suitable domains of *values* V and *results* R, respectively. These have to be large enough to contain all possible denotations of expressions and computations, which is usually achieved by solving suitable recursive domain equations, such as

$$V = \mathbb{N}_\bot \oplus \{0,1\}_\bot \oplus \{\star\}_\bot \oplus \mathbb{I}_\bot \oplus R^V \oplus R^R,$$
$$R = (V + \mathbb{I} \times \mathbb{O} \times V \times R^V)_\bot.$$

The summands in the equation for V correspond to various expression types. The recursive domain equation for R says that a non-bottom element of R is either a value, or a quadruple $(\iota, \mathtt{op}, v, \kappa)$ corresponding to the operation $\iota \,\#\, \mathtt{op}$ applied to parameter v and with continuation κ. However, as in [17], we shall assume only that V and R are large enough to contain the various components needed for the semantics as *retracts*:

$$\mathbb{N}_\bot \underset{\mathsf{r}_{\mathtt{nat}}}{\overset{\mathsf{s}_{\mathtt{nat}}}{\rightleftarrows}} V \qquad \{0,1\}_\bot \underset{\mathsf{r}_{\mathtt{bool}}}{\overset{\mathsf{s}_{\mathtt{bool}}}{\rightleftarrows}} V \qquad \{\star\}_\bot \underset{\mathsf{r}_{\mathtt{unit}}}{\overset{\mathsf{s}_{\mathtt{unit}}}{\rightleftarrows}} V$$

$$\mathbb{I}_\bot \underset{\mathsf{r}_{\mathtt{effect}}}{\overset{\mathsf{s}_{\mathtt{effect}}}{\rightleftarrows}} V \qquad R^V \underset{\mathsf{r}_{\to}}{\overset{\mathsf{s}_{\to}}{\rightleftarrows}} V \qquad R^R \underset{\mathsf{r}_{\Rightarrow}}{\overset{\mathsf{s}_{\Rightarrow}}{\rightleftarrows}} V$$

and

$$(V + \mathbb{I} \times \mathbb{O} \times V \times R^V)_\bot \underset{\mathsf{r}_{\mathtt{res}}}{\overset{\mathsf{s}_{\mathtt{res}}}{\rightleftarrows}} R$$

Thus, R and V could be solutions to the above domain equations, but they could also both be a universal domain, or just a (non-trivial) reflexive domain.[1] Note that there are further canonical retractions

$$V \underset{\mathsf{r}_{\mathtt{val}}}{\overset{\mathsf{s}_{\mathtt{val}}}{\rightleftarrows}} (V + \mathbb{I} \times \mathbb{O} \times V \times R^V)_\bot$$

$$(\mathbb{I} \times \mathbb{O} \times V \times R^V)_\bot \underset{\mathsf{r}_{\mathtt{oper}}}{\overset{\mathsf{s}_{\mathtt{oper}}}{\rightleftarrows}} (V + \mathbb{I} \times \mathbb{O} \times V \times R^V)_\bot$$

which in combination with the section-retraction pair $(\mathsf{s}_{\mathtt{res}}, \mathsf{r}_{\mathtt{res}})$ convert elements of R to pure values and operations, respectively.

The domain $(V + \mathbb{I} \times \mathbb{O} \times V \times R^V)_\bot$ has the following *recursion principle*. Given a domain D and maps

$$h_{\mathtt{val}} : V \longrightarrow D \qquad \text{and} \qquad h_{\mathtt{oper}} : \mathbb{I} \times \mathbb{O} \times V \times R^V \longrightarrow D,$$

there is a unique strict map $h : (V + \mathbb{I} \times \mathbb{O} \times V \times R^V)_\bot \longrightarrow D$ such that, for all $v \in V$, $\iota \in \mathbb{I}$, $\mathtt{op} \in \mathbb{O}$, $\kappa : V \to R$,

$$h(\mathsf{s}_{\mathtt{val}}(v)) = h_{\mathtt{val}}\,(v),$$
$$h(\mathsf{s}_{\mathtt{oper}}(\iota, \mathtt{op}, v, \kappa)) = h_{\mathtt{oper}}(\iota, \mathtt{op}, v, h \circ \mathsf{r}_{\mathtt{res}} \circ \kappa).$$

[1] A domain D is *reflexive* if it contains its functions space D^D as a retract, and is therefore a model of the untyped λ-calculus.

Moreover, h depends continuously on the data $h_{\mathtt{val}}$ and $h_{\mathtt{oper}}$. For example, given a map $f : V \to R$, there is a unique strict map

$$f^\dagger : (V + \mathbb{I} \times \mathbb{O} \times V \times R^V)_\bot \longrightarrow R,$$

called the *lifting* of f, which depends on f continuously and satisfies the recursive equations

$$\begin{aligned} f^\dagger(\mathsf{s}_{\mathtt{val}}(v)) &= f(v), \\ f^\dagger(\mathsf{s}_{\mathtt{oper}}(\iota, \mathtt{op}, v, \kappa)) &= \mathsf{s}_{\mathtt{oper}}(\iota, \mathtt{op}, v, f^\dagger \circ \mathsf{r}_{\mathtt{res}} \circ \kappa). \end{aligned}$$

An *environment* η is a map from variable names to values. We denote by $\eta[x \mapsto v]$ the environment which assigns v to x and otherwise behaves as η. The untyped denotational semantics of expressions assigns to each expression a map from environments to V, as follows. First we have the standard constructs:

$$\begin{aligned} [\![x]\!]\,\eta &= \eta(x) \\ [\![()]\!]\,\eta &= \mathsf{s}_{\mathtt{unit}}(\star) \\ [\![0]\!]\,\eta &= \mathsf{s}_{\mathtt{nat}}(0) \\ [\![\mathtt{succ}\ e]\!]\,\eta &= \begin{cases} \mathsf{s}_{\mathtt{nat}}(n+1) & \text{if } \mathsf{r}_{\mathtt{nat}}([\![e]\!]\,\eta) = n \in \mathbb{N}, \\ \mathsf{s}_{\mathtt{nat}}(\bot) & \text{if } \mathsf{r}_{\mathtt{nat}}([\![e]\!]\,\eta) = \bot \end{cases} \\ [\![\mathtt{true}]\!]\,\eta &= \mathsf{s}_{\mathtt{bool}}(1) \\ [\![\mathtt{false}]\!]\,\eta &= \mathsf{s}_{\mathtt{bool}}(0) \\ [\![\iota]\!]\,\eta &= \mathsf{s}_{\mathtt{effect}}(\iota) \\ [\![\mathtt{fun}\ x \mapsto c]\!]\,\eta &= \mathsf{s}_{\to}(\lambda v : V.\, [\![c]\!]\,(\eta[x \mapsto v])) \end{aligned}$$

When e evaluates to an instance we interpret $e\,\mathtt{\#\,op}$ as a generic effect:

$$[\![e\,\mathtt{\#\,op}]\!]\,\eta = \begin{cases} \mathsf{s}_{\to}(\lambda v : V.\, \mathsf{s}_{\mathtt{res}}(\mathsf{s}_{\mathtt{oper}}(\iota, \mathtt{op}, v, \mathsf{s}_{\mathtt{res}} \circ \mathsf{s}_{\mathtt{val}}))) & \text{if } \mathsf{r}_{\mathtt{effect}}([\![e]\!]\,\eta) = \iota \in \mathbb{I}, \\ \mathsf{s}_{\to}(\lambda v : V.\, \mathsf{s}_{\mathtt{res}}(\bot)) & \text{if } \mathsf{r}_{\mathtt{effect}}([\![e]\!]\,\eta) = \bot. \end{cases}$$

The interpretation of a handler is

$$[\![\mathtt{handler\ val}\ x \mapsto c_v \mid (e_i\,\mathtt{\#\,op}_i\ x_i\ k_i \mapsto c_i)_i]\!]\,\eta = \mathsf{s}_{\Rightarrow}(h \circ \mathsf{r}_{\mathtt{res}})$$

where $h : (V + \mathbb{I} \times \mathbb{O} \times V \times R^V)_\bot \to R$ is characterized using the recursion principle for R as follows:

1. if one of the $\mathsf{r}_{\mathtt{effect}}([\![e_i]\!]\,\eta)$ is $\bot$ we set $h = \lambda x.\, \mathsf{s}_{\mathtt{res}}(\bot)$, otherwise
2. if $\mathsf{r}_{\mathtt{effect}}([\![e_i]\!]\,\eta) = \iota_i \in \mathbb{I}$ for all i, then we define h by cases as

$$\begin{aligned} h(\mathsf{s}_{\mathtt{val}}(v)) &= [\![c_v]\!]\,(\eta[x \mapsto v]) \\ h(\mathsf{s}_{\mathtt{oper}}(\iota_i, \mathtt{op}_i, v, \kappa)) &= [\![c_i]\!]\,(\eta[x_i \mapsto v, k_i \mapsto h \circ \mathsf{r}_{\mathtt{res}} \circ \kappa]) \\ &\qquad \text{for all } i, \\ h(\mathsf{s}_{\mathtt{oper}}(\iota, \mathtt{op}, v, \kappa)) &= \mathsf{s}_{\mathtt{res}}(\mathsf{s}_{\mathtt{oper}}(\iota, \mathtt{op}, v, h \circ \mathsf{r}_{\mathtt{res}} \circ \kappa)) \\ &\qquad \text{if } (\iota, \mathtt{op}) \neq (\iota_i, \mathtt{op}_i) \text{ for all } i. \end{aligned}$$

There may be overlapping cases in the second clause, i.e., it could happen that $(\iota_i, \mathtt{op}_i) = (\iota_j, \mathtt{op}_j)$, in which case the clause that is listed first counts.

Computations are modeled as maps from environments to results. Promotion of expressions to computations is just the inclusion of V into R,

$$[\![\mathtt{val}\ e]\!]\,\eta = \mathsf{s}_{\mathtt{res}}(\mathsf{s}_{\mathtt{val}}([\![e]\!]\,\eta)).$$

The `let` statement corresponds to monadic-style binding:

$$[\![\mathtt{let}\ x = c_1\ \mathtt{in}\ c_2]\!]\,\eta = (\lambda v : V.\,[\![c_2]\!]\,(\eta[x \mapsto v]))^{\dagger}(\mathsf{r}_{\mathtt{res}}([\![c_1]\!]\,\eta)),$$

A recursive function definition is interpreted as

$$[\![\mathtt{let\ rec}\ f\ x = c_1\ \mathtt{in}\ c_2]\!]\,\eta = [\![c_2]\!]\,(\eta[f \mapsto \mathsf{s}_{\to}(t)])$$

where $t : V \to R$ is the least fixed point of the map

$$t \mapsto (\lambda v : V.\,[\![c_1]\!]\,(\eta[f \mapsto \mathsf{s}_{\to}(t), x \mapsto v])).$$

The elimination forms are interpreted in the usual way as:

$$[\![\mathtt{iszero}\ e]\!]\,\eta = \begin{cases} \mathsf{s}_{\mathtt{res}}(\mathsf{s}_{\mathtt{val}}(\mathsf{s}_{\mathtt{bool}}(1))) & \text{if } 0 = \mathsf{r}_{\mathtt{nat}}([\![e]\!]\,\eta) \\ \mathsf{s}_{\mathtt{res}}(\mathsf{s}_{\mathtt{val}}(\mathsf{s}_{\mathtt{bool}}(0))) & \text{if } 0 \neq \mathsf{r}_{\mathtt{nat}}([\![e]\!]\,\eta) \in \mathbb{N} \\ \mathsf{s}_{\mathtt{res}}(\bot) & \text{if } \bot = \mathsf{r}_{\mathtt{nat}}([\![e]\!]\,\eta) \end{cases}$$

$$[\![\mathtt{if}\ e\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2]\!]\,\eta = \begin{cases} [\![c_1]\!]\,\eta & \text{if } \mathsf{r}_{\mathtt{bool}}([\![e]\!]\,\eta) = 1 \\ [\![c_2]\!]\,\eta & \text{if } \mathsf{r}_{\mathtt{bool}}([\![e]\!]\,\eta) = 0 \\ \mathsf{s}_{\mathtt{res}}(\bot) & \text{otherwise} \end{cases}$$

$$[\![\mathtt{absurd}\ e]\!]\,\eta = \mathsf{s}_{\mathtt{res}}(\bot)$$

$$[\![e_1\ e_2]\!]\,\eta = \mathsf{r}_{\to}([\![e_1]\!]\,\eta)([\![e_2]\!]\,\eta)$$

Finally, the handling construct is just an application

$$[\![\mathtt{with}\ e\ \mathtt{handle}\ c]\!]\,\eta = \mathsf{r}_{\Rightarrow}([\![e]\!]\,\eta)([\![c]\!]\,\eta).$$

The denotational semantics of expressions and computations is written in such a way that it immediately suggests a big-step operational semantics. Indeed, in the implementation of Eff the main evaluation function is essentially a transcription of the above rules into OCaml [11].

3.2 Semantics of Types

According to John Reynolds, the untyped semantics of terms given in §3.1 is related to a semantics of types by specifying, for all expression types A and computation types $\underline{C}$, section-retraction pairs

$$[\![A]\!] \underset{\mathsf{r}_A}{\overset{\mathsf{s}_A}{\rightleftarrows}} V \qquad \text{and} \qquad [\![\underline{C}]\!] \underset{\mathsf{r}_{\underline{C}}}{\overset{\mathsf{s}_{\underline{C}}}{\rightleftarrows}} R\ ,$$

A value $v \in V$ and a result $r \in R$ have types A and $\underline{C}$, respectively, if they are members of the corresponding retracts, which is expressed by fix-point equations

$$v = \mathsf{s}_A(\mathsf{r}_A(v)) \qquad \text{and} \qquad r = \mathsf{s}_{\underline{C}}(\mathsf{r}_{\underline{C}}(r)).$$

Such semantics works well for the *structural* part of types, i.e., everything except the effect system. Therefore, for the effect system we are going to use a second level of semantics by equipping the retracts with *partial equivalence relations (per)*.[2] This way the semantics is neatly stratified: we may ignore the effect system and use only the domain-theoretic part of semantics to get a more traditional account of the language, or we also include the effect system and the pers for a more refined meaning of types.

For the basic types, the effect types and the function type we set

$$\begin{aligned} [\![\mathtt{nat}]\!] &= \mathbb{N}_\bot, & [\![\mathtt{bool}]\!] &= \{0,1\}_\bot, \\ [\![\mathtt{unit}]\!] &= \{\star\}_\bot, & [\![\mathtt{empty}]\!] &= \varnothing_\bot, \\ [\![E^\rho]\!] &= \mathbb{I}_\bot, & [\![A \to \underline{C}]\!] &= [\![\underline{C}]\!]^{[\![A]\!]}, \end{aligned}$$

The section-retraction pairs for the basic types and the effect types are the ones given previously, and for the function type we set

$$\mathsf{s}_{A\to\underline{C}}(f) = \mathsf{s}_\to(\mathsf{s}_{\underline{C}} \circ f \circ \mathsf{r}_A) \qquad \text{and} \qquad \mathsf{r}_{A\to\underline{C}}(v) = \mathsf{r}_{\underline{C}} \circ \mathsf{r}_\to(v) \circ \mathsf{s}_A.$$

The pers for the basic types `nat`, `bool`, `unit`, and `empty` are identities on the *total* elements, while the per for an effect type E^ρ is identity restricted to ρ:

$$\begin{aligned} v \sim_{\mathtt{nat}} v' &\iff v = v' \in \mathbb{N}, & v \sim_{\mathtt{bool}} v' &\iff v = v' \in \{0,1\}, \\ v \sim_{\mathtt{unit}} v' &\iff v = v' = \star, & v \sim_{\mathtt{empty}} v' &\iff \bot, \\ v \sim_{E^\rho} v' &\iff v = v' \in \rho. \end{aligned}$$

The function type $A \to \underline{C}$ has the standard per

$$f \sim_{A\to\underline{C}} f' \iff \forall v, v' \in [\![A]\!] \,.\, (v \sim_A v' \implies f(v) \sim_{\underline{C}} f'(v')).$$

Similarly, handler types $\underline{C} \Rightarrow \underline{D}$ are treated as functions from computations to computations. The underlying domain is the function space domain with the section-retraction pair

$$[\![\underline{C} \Rightarrow \underline{D}]\!] = [\![\underline{D}]\!]^{[\![\underline{C}]\!]} \qquad \text{and} \qquad [\![\underline{C} \Rightarrow \underline{D}]\!] \underset{\mathsf{r}_{\underline{C}\Rightarrow\underline{D}}}{\overset{\mathsf{s}_{\underline{C}\Rightarrow\underline{D}}}{\rightleftarrows}} V \ ,$$

defined in the expected way,

$$\mathsf{s}_{\underline{C}\Rightarrow\underline{D}}(h) = \mathsf{s}_\Rightarrow(\mathsf{s}_{\underline{D}} \circ h \circ \mathsf{r}_{\underline{C}}) \qquad \text{and} \qquad \mathsf{r}_{\underline{C}\Rightarrow\underline{D}}(v) = \mathsf{r}_{\underline{D}} \circ \mathsf{r}_\Rightarrow(v) \circ \mathsf{s}_{\underline{C}}.$$

Also the per is the one for function types,

$$h \sim_{\underline{C}\Rightarrow\underline{D}} h' \iff \forall r, r' \in [\![\underline{C}]\!] \,.\, (r \sim_{\underline{C}} r' \Rightarrow h(r) \sim_{\underline{D}} h'(r')).$$

[2] Recall that a per is a symmetric and transitive relation.

Semantics of computation types is a bit more involved. Let $\mathbb{D}$ be the set of all pairs $(\iota, \mathtt{op})$ where ι is an instance whose associated effect type is E and $\mathtt{op} : A_{\mathtt{op}} \to B_{\mathtt{op}}$ is an operation declared by E. The domain associated with the computation type $A!\delta$ is the solution of recursive domain equation

$$[\![A!\delta]\!] = \big([\![A]\!] + \coprod_{(\iota,\mathtt{op})\in\mathbb{D}} [\![A_{\mathtt{op}}]\!] \times [\![A!\delta]\!]^{[\![B_{\mathtt{op}}]\!]}\big)_\perp.$$

Because we want to keep the domain-theoretic part of semantics independent of the effect system, we do not use the information provided by δ, and so we have to sum over *all* operations in $\mathbb{D}$. We shall encode information about δ in the per $\sim_{A!\delta}$. There are canonical retractions

$$[\![A]\!] \underset{\mathsf{r}^{A!\delta}_{\mathtt{val}}}{\overset{\mathsf{s}^{A!\delta}_{\mathtt{val}}}{\rightleftarrows}} [\![A!\delta]\!] \quad \text{and} \quad \big(\coprod_{(\iota,\mathtt{op})\in\mathbb{D}} [\![A_{\mathtt{op}}]\!] \times [\![A!\delta]\!]^{[\![B_{\mathtt{op}}]\!]}\big)_\perp \underset{\mathsf{r}^{A!\delta}_{\mathtt{oper}}}{\overset{\mathsf{s}^{A!\delta}_{\mathtt{oper}}}{\rightleftarrows}} [\![A!\delta]\!]$$

Just like for R, we shall not use the domain equation for $[\![A!\delta]\!]$, but rather only the above section-retraction pairs. The retraction $\mathsf{r}_{A!\delta} : R \to [\![A!\delta]\!]$ is the composition of $\mathsf{r}_{\mathtt{res}}$ and the map $f : (V + \mathbb{I} \times \mathbb{O} \times V \times R^V)_\perp \longrightarrow [\![A!\delta]\!]$ defined by the recursion principle as

$$\begin{aligned} f(\mathsf{s}_{\mathtt{val}}\,(v)) &= \mathsf{s}^{A!\delta}_{\mathtt{val}}(\mathsf{r}_A(v)), \\ f(\mathsf{s}_{\mathtt{oper}}(\iota, \mathtt{op}, v, \kappa)) &= \begin{cases} \mathsf{s}^{A!\delta}_{\mathtt{oper}}(\iota, \mathtt{op}, \mathsf{r}_{A_{\mathtt{op}}}(v), f \circ \mathsf{r}_{\mathtt{res}} \circ \kappa \circ \mathsf{s}_{B_{\mathtt{op}}}) & \text{for } (\iota, \mathtt{op}) \in \mathbb{D}, \\ \perp & \text{for } (\iota, \mathtt{op}) \notin \mathbb{D}. \end{cases} \end{aligned}$$

The section $\mathsf{s}_{A!\delta}$ is defined similarly, using an analogous recursion principle for

$$\big([\![A]\!] + \coprod_{(\iota,\mathtt{op})\in\mathbb{D}} [\![A_{\mathtt{op}}]\!] \times [\![A!\delta]\!]^{[\![B_{\mathtt{op}}]\!]}\big)_\perp.$$

The per $\sim_{A!\delta}$ is defined inductively as the least one satisfying:

1. $\perp \sim_{A!\delta} \perp$,
2. for all $v, v' \in [\![A]\!]$, if $v \sim_A v'$ then $\mathsf{s}^{A!\delta}_{\mathtt{val}}(v) \sim_{A!\delta} \mathsf{s}^{A!\delta}_{\mathtt{val}}(v')$,
3. for all $(\iota \,\#\, \mathtt{op}) \in \delta$, all $v, v' \in [\![A_{\mathtt{op}}]\!]$, and all $\kappa, \kappa' \in [\![B_{\mathtt{op}} \to A!\delta]\!]$,

$$v \sim_{A_{\mathtt{op}}} v' \wedge \kappa \sim_{B_{\mathtt{op}} \to A!\delta} \kappa' \implies \mathsf{s}^{A!\delta}_{\mathtt{oper}}(\iota, \mathtt{op}, v, \kappa) \sim_{A!\delta} \mathsf{s}^{A!\delta}_{\mathtt{oper}}(\iota, \mathtt{op}, v', \kappa').$$

The pers $\sim_A$ and $\sim_{\underline{C}}$ are defined on the corresponding domains $[\![A]\!]$ and $[\![\underline{C}]\!]$. We can transfer them along sections to pers $\approx_A$ and $\approx_{\underline{C}}$ on V and R by

$$v \sim_A v' \iff \mathsf{s}_A(v) \approx_A \mathsf{s}_A(v') \qquad \text{and} \qquad r \sim_{\underline{C}} r' \iff \mathsf{s}_{\underline{C}}(r) \approx_{\underline{C}} \mathsf{s}_{\underline{C}}(r').$$

Note that $v \approx_A v$ implies $\mathsf{s}_A(\mathsf{r}_A(v)) = v$, and similarly for $\approx_{\underline{C}}$.

We connect the untyped semantics of terms and the semantics of types with a soundness theorem.

Theorem 1 (Matching semantics of terms and types). *Let Γ be a typing context and η an environment such that, for every $x_i : A_i$ in Γ, $\eta(x_i) \approx_{A_i} \eta(x_i)$.*

1. *If* $\Gamma \vdash e : A$ *then* $[\![e]\!]\,\eta \approx_A [\![e]\!]\,\eta$.
2. *If* $\Gamma \vdash c : \underline{C}$ *then* $[\![c]\!]\,\eta \approx_{\underline{C}} [\![c]\!]\,\eta$.
3. *If* $A \leq A'$ *then* $[\![A]\!] = [\![A']\!]$ *and* $(\sim_A) \subseteq (\sim_{A'})$.
4. *If* $\underline{C} \leq \underline{C}'$ *then* $[\![\underline{C}]\!] = [\![\underline{C}']\!]$ *and* $(\sim_{\underline{C}}) \subseteq (\sim_{\underline{C}'})$.

Proof. The proof proceeds by induction on the judgment derivations. The typing rule for recursive function definitions works because all the pers for computation types are *admissible*, i.e., they contain the least element and are closed under suprema of chains. Likewise, the rule for elimination of the empty type works because the pers for computation types contain the least element. □

3.3 Equational Reasoning

We can use the denotational semantics to validate program transformations. This is all very familiar, so we just review the main idea. Consider computations c_1 and c_2 which have type $\underline{C}$ in the typing context Γ. Say that c_1 and c_2 are *(semantically) equivalent* and write $c_1 \equiv c_2$ when, for any environment η such that $\eta(x_i) \approx_{A_i} \eta(x_i)$ for all $(x_i : A_i) \in \Gamma$, we have

$$[\![c_1]\!]\,\eta \approx_{\underline{C}} [\![c_2]\!]\,\eta.$$

A similar definition can be made for equivalence of expressions. Then $\equiv$ is an equivalence relation which satisfies the "substitution of equals" principle, i.e., for a well-typed evaluation context $\mathcal{C}[-]$, $c_1 \equiv c_2$ implies $\mathcal{C}[c_1] \equiv \mathcal{C}[c_2]$, and similarly for expressions. Therefore, we may safely replace a computation or an expression with an equivalent one. In fact, since evaluation is a form of program transformation, we have a criterion for correctness of implementation: an evaluation strategy is *correct* with respect to the semantics if it preserves semantic equivalence.

We list the fundamental equivalences which form the basis of an evaluation strategy, and they are also useful for equational reasoning about effectful computations. First, we have the equivalences governing handlers that were informally explained in §2.2. If h is the handler $\texttt{handler}\ \texttt{val}\ x \mapsto c_v \mid (\iota_i\ \#\ \texttt{op}_i\ x_i\ k_i \mapsto c_i)_i$ then

$$\texttt{with}\ h\ \texttt{handle}\ (\texttt{val}\ e) \quad \equiv \quad c_v[x \mapsto e]$$

and

$$\texttt{with}\ h\ \texttt{handle}\ (\texttt{let}\ y = \iota_i\ \#\ \texttt{op}_i\ e\ \texttt{in}\ c) \quad \equiv \quad c_i[x_i \mapsto e, k_i \mapsto (\texttt{fun}\ y \mapsto \texttt{with}\ h\ \texttt{handle}\ c)]$$

and, assuming $\iota \neq \iota_i$ for all i,

$$\texttt{with}\ h\ \texttt{handle}\ (\texttt{let}\ y = \iota\ \#\ \texttt{op}\ e\ \texttt{in}\ c) \quad \equiv \quad \texttt{let}\ y = \iota\ \#\ \texttt{op}\ e\ \texttt{in}\ (\texttt{with}\ h\ \texttt{handle}\ c).$$

These equations were identified before by [8]. A variety of other equivalences is readily validated, such as $\beta\eta$-conversions for functions and the "associativity" law [18] (y must not occur freely in c_3)

$$\texttt{let}\ x = (\texttt{let}\ y = c_1\ \texttt{in}\ c_2)\ \texttt{in}\ c_3 \quad \equiv \quad \texttt{let}\ y = c_1\ \texttt{in}\ (\texttt{let}\ x = c_2\ \texttt{in}\ c_3).$$

The proof of this law proceeds by induction, as it uses the inductive nature of the per associated with the type of c_1. The inductive nature of the argument is similar to the one given by [19].

4 Discussion

We have presented a fairly simple effect system, which nevertheless allows us to deduce non-trivial properties of computations. There are several aspects of the system which we would like and plan to improve.

First, the per model is clearly suggesting that parametric polymorphism should blend naturally with the effect system. Once this is done we may have to pass to a *parametric* rather than the naive per model.

Second, in full Eff instances may be created dynamically with `new` E. To account for these in semantics, we would have to further complicate both the effect system and the semantics so that they could account for *freshness* phenomena. This would presumably follow the work of [20,21].

Third, we wrote the typing rules so that they are suitable for effect checking, but we really want effect inference. Initial experiments and a prototype implementation suggest that this should be not only doable, but quite likely useful too.

References

1. Lucassen, J.M., Gifford, D.K.: Polymorphic effect systems. In: Proceedings of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 1988, pp. 47–57. ACM, New York (1988)
2. Kammar, O., Plotkin, G.D.: Algebraic foundations for effect-dependent optimisations. In: Field, J., Hicks, M. (eds.) POPL, pp. 349–360. ACM (2012)
3. Talpin, J.P., Jouvelot, P.: Polymorphic type, region and effect inference. Journal of Functional Programming 2(3), 245–271 (1992)
4. Wadler, P.: The marriage of effects and monads. ACM SIGPLAN Notices 34(1), 63–74 (1999)
5. Kammar, O., Lindley, S., Oury, N.: Handlers in action. In: The 1st ACM SIGPLAN Workshop on Higher-Order Programming with Effects (2012)
6. Bauer, A., Pretnar, M.: Programming with algebraic effects and handlers. arXiv:1203.1539 (2012)
7. Plotkin, G., Power, J.: Algebraic operations and generic effects. Applied Categorical Structures 11(1), 69–94 (2003)
8. Plotkin, G., Pretnar, M.: Handlers of algebraic effects. In: Castagna, G. (ed.) ESOP 2009. LNCS, vol. 5502, pp. 80–94. Springer, Heidelberg (2009)

9. Kiselyov, O.: Lazy IO breaks equational reasoning (April 2013), `http://okmij.org/ftp/Haskell/index.html#lazyIO-not-True`
10. Milner, R., Tofte, M., Harper, R., MacQueen, D.: The Definition of Standard ML. MIT Press (1997)
11. Leroy, X., Doligez, D., Frisch, A., Garrigue, J., Rémy, D., Vouillon, J.: The OCaml system (release 3.12): Documentation and user's manual. Institut National de Recherche en Informatique et en Automatique (2011)
12. Benton, N., Hughes, J., Moggi, E.: Monads and effects. In: Barthe, G., Dybjer, P., Pinto, L., Saraiva, J. (eds.) APPSEM 2000. LNCS, vol. 2395, pp. 42–122. Springer, Heidelberg (2002)
13. Levy, P.B., Power, J., Thielecke, H.: Modelling environments in call-by-value programming languages. Information and Computation 185, 182–210 (2003)
14. Fuh, Y.C., Mishra, P.: Type inference with subtypes. In: Ganzinger, H. (ed.) ESOP 1988. LNCS, vol. 300, pp. 94–114. Springer, Heidelberg (1988)
15. Hyland, M., Plotkin, G., Power, J.: Combining effects: Sum and tensor. Theoretical Computer Science 357(1-3), 70–99 (2006)
16. Plotkin, G., Power, J.: Notions of computation determine monads. In: Nielsen, M., Engberg, U. (eds.) FOSSACS 2002. LNCS, vol. 2303, pp. 342–356. Springer, Heidelberg (2002)
17. Reynolds, J.: The meaning of types—from intrinsic to extrinsic semantics. Technical report, Department of Computer Science, University of Aarhus (2000)
18. Moggi, E.: Notions of computation and monads. Information and Computation 93(1), 55–92 (1991)
19. Pretnar, M.: The Logic and Handling of Algebraic Effects. PhD thesis, School of Informatics, University of Edinburgh (2010)
20. Murdoch, G.J., Pitts, A.M.: A new approach to abstract syntax with variable binding. Formal Aspects of Computing 13(3-5), 341–363 (2001)
21. Stark, I.: Categorical models for local names. LISP and Symbolic Computation 9(1), 77–107 (1996)

Automata and Algebras for Infinite Words and Trees

Mikołaj Bojańczyk*

University of Warsaw

Regular languages can be studied not just for finite words, but also for finite trees, infinite words and infinite trees. Almost all of the theory of regular languages, such as closure under boolean operations, works also for these extensions, but the constructions are significantly more challenging and mathematically interesting. For instance, automata for infinite words can be determinised under a suitable choice of acceptance condition, but the proof requires an intricate combinatorial construction.

In the first part of my talk, I will describe the now classical results on automata recognising regular languages of infinite words and infinite trees. I will mention the connection of automata with monadic second-order logic, discovered by Büchi and Rabin, which has been one of the main sources of inspiration in the theory of automata for infinite objects. More on these topics can be found in the general survey [4] or in the collection of more specialised surveys [1].

The second part of my talk will be on the algebraic approach to regular languages. For finite words, the algebraic approach is to use semigroups instead of automata. The beauty of the algebraic approach is that it uncovers connections between classical mathematical concepts and formal language theory. For instance, a celebrated theorem of Schützenberger says that a regular language of finite words can be defined by a regular expression without the star (but with complementation) if and only if the language can be recognised by a semigroup that does not contain any nontrivial group. There is also a well understood algebraic theory for infinite words, which uses variants of semigroups. The algebraic theory for regular languages of finite words is described in the book [3], and its extensions to infinite words are described in the book [2].

However, the algebraic theory of infinite trees, or even finite trees, is currently not well understood, and seems to be a very challenging problem. I will end my talk by describing this problem, and why it might be interesting for people studying algebra and coalgebra.

References

1. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games: A Guide to Current Research. LNCS, vol. 2500. Springer, Heidelberg (2002)

* Author supported by ERC Starting Grant "Sosna".

R. Heckel and S. Milius (Eds.): CALCO 2013, LNCS 8089, pp. 17–18, 2013.

2. Perrin, D., Pin, J.-É.: Infinite Words. Elsevier (2004)
3. Straubing, H.: Finite Automata, Formal Languages, and Circuit Complexity. Birkhäuser (1994)
4. Thomas, W.: Languages, automata, and logic. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. III, pp. 389–455. Springer, New York (1997)

Positive Inductive-Recursive Definitions

Neil Ghani[1], Lorenzo Malatesta[1], and Fredrik Nordvall Forsberg[2]

[1] University of Strathclyde, UK
[2] Swansea University, UK

Abstract. We introduce a new theory of data types which allows for the definition of data types as initial algebras of certain functors $\mathsf{Fam}\,\mathbb{C} \to \mathsf{Fam}\,\mathbb{C}$. This theory, which we call *positive inductive-recursive definitions*, is a generalisation of Dybjer and Setzer's theory of inductive-recursive definitions within which $\mathbb{C}$ had to be discrete – our work can therefore be seen as lifting this restriction. This is a substantial endeavour as we need to not only introduce a type of codes for such data types (as in Dybjer and Setzer's work), but also a type of morphisms between such codes (which was not needed in Dybjer and Setzer's development). We show how these codes are interpreted as functors on $\mathsf{Fam}\,\mathbb{C}$ and how these morphisms of codes are interpreted as natural transformations between such functors. We then give an application of positive inductive-recursive definitions to the theory of nested data types. Finally we justify the existence of positive inductive-recursive definitions by adapting Dybjer and Setzer's set-theoretic model to our setting.

1 Introduction

Inductive types are the bricks of a dependently typed programming language: they represent the building blocks on which any other type is built. The mortar the dependently typed programmer has at her disposal for computation with dependent types is recursion. Usually, a type A is defined inductively, and then terms or types can be defined recursively over the structure of A. The theory of inductive-recursive definitions [7,8] explores the simultaneous combination of these two basic ingredients, pushing the limits of the theoretical foundations of data types.

The key example of an inductive-recursive definition is Martin-Löf's universe à la Tarski [19]. A type U consisting of codes for small types is introduced, together with a decoding function T, which maps codes to the types they denote. The definition is both inductive and recursive; the type U is defined inductively, and the decoding function T is defined recursively on the way the elements of U are generated. The definition needs to be simultaneous, since the introduction rules for U refer to T. We illustrate this by means of a concrete example: say we want to define a data type representing a universe containing a name for the natural numbers, closed under Σ-types. Such a universe will be the smallest family of sets (U, T) satisfying the following equations

R. Heckel and S. Milius (Eds.): CALCO 2013, LNCS 8089, pp. 19–33, 2013.

$$\begin{aligned} U &= 1 \;+\; \Sigma\, u{:}U.\, Tu \to U \\ T(\mathsf{inl}\ *) &= \mathbb{N} \\ T(\mathsf{inr}\ (u, f)) &= \Sigma x{:}Tu.\, T(fx) \end{aligned} \tag{1}$$

In this definition we see how ground types and the type constructor Σ are reflected in U. The left summand of the right hand side of the equation defining U is a code for natural numbers, while the right summand is a code reflecting Σ-types. Indeed the name of a Σ-type, $\Sigma\, A\, B$ for $A : \mathsf{Set}$, $B : A \to \mathsf{Set}$, in the universe (U, T) will consists of a name in U for the type A, i.e. an element $u : U$, and a function $f : Tu \to U$ representing the A-indexed family of sets B. The decoding function T maps elements of U according to the description above: the code for natural numbers decodes to the set of natural numbers $\mathbb{N}$ while an element (u, f) of the right summand decodes to the Σ-type it denotes. Other examples of inductive-recursive definitions have also appeared in the literature, such as e.g. Martin-Löf's computability predicates [18] or Aczel's Frege structures [3]. Lately the use of inductive-recursive definitions to encode invariants in ordinary data structures has also been considered [11].

Dybjer's [7] insight was that these examples are instances of a general notion, which Dybjer and Setzer [8] later found a finite axiomatisation of. Their theory of inductive-recursive definitions IR consists of: (i) a representation of types as initial algebras of functors; (ii) a grammar for defining such functors. Elements of the grammar are called IR codes, while functors associated to IR codes are called IR functors. The theory naturally covers simpler inductive types such as lists, trees, vectors, red-black trees etc. as well. Dybjer and Setzer [9] then gave an initial algebra semantics for IR codes by showing that IR functors are naturally defined on the category $\mathsf{Fam}\, D$ of families of elements of a (possibly large) type D and that these functors do indeed have initial algebras. More generally, abstracting on the families construction and the underlying families fibration $\pi : \mathsf{Fam}\, D \to \mathsf{Set}$, we have recently shown how to interpret IR functors in an arbitrary fibration endowed with the appropriate structure [15]. In this article, we will only consider the families fibration.

There is, however, a complication. When interpreting IR functors such as those building universes closed under dependent products, the mixture of covariance and contravariance intrinsic in the Π operator forces one to confine attention to functors $\mathsf{Fam}\,|\mathbb{C}| \to \mathsf{Fam}\,|\mathbb{C}|$ or, equivalently, to work with only those morphisms between families which are commuting triangles. As we have shown [15], more abstractly, this corresponds to working in the split cartesian fragment of the families fibration $\pi : \mathsf{Fam}\,\mathbb{C} \to \mathsf{Set}$, i.e. to only consider those morphisms in $\mathsf{Fam}\,\mathbb{C}$ which represent strict reindexing. In this paper we remove this constraint and hence explore a further generalization of IR, orthogonal to the one proposed in Ghani et al. [15]. We investigate the necessary changes of IR needed to provide a class of codes which can be interpreted as functors $\mathsf{Fam}\,\mathbb{C} \to \mathsf{Fam}\,\mathbb{C}$. This leads us to consider a new variation IR^+ of inductive-recursive definitions which we call *positive inductive-recursive definitions*. The most substantial aspect of this

new theory is that in order to define these new codes, one needs also to define the morphisms between those codes. This is no handle-turning exercise!

We first recall Dybjer and Setzer's theory of inductive-recursive definitions (Section 2). To develop the theory we then (i) introduce a syntax and semantics consisting of IR^+ codes and their morphisms, and an explanation how these codes are interpreted as functors $\mathsf{Fam}\,\mathbb{C} \to \mathsf{Fam}\,\mathbb{C}$, where $\mathbb{C}$ is an arbitrary category (Section 3); (ii) use positive inductive-recursive definitions to shed new light on nested data types (Section 4); (iii) give a detailed comparison with the existing theory of IR (Section 5); (iv) adapt Dybjer and Setzer's model construction to our setting (Section 6).

The paper uses a mixture of categorical and type theoretic constructions. However, the reader should bear in mind that the foundations of this paper are type theoretic. In other words, all constructions should be understood to take place in extensional Martin-Löf type theory with one universe Set. This is entirely standard in the literature. The one exception is the use of a Mahlo cardinal required to prove that positive inductive recursive functors have initial algebras in Section 6. It should be emphasised that the Mahlo cardinal is only used to justify the soundness of the theory, and does not play any computational role. We refer the interested reader to Dybjer and Setzer [8] – they use a Mahlo cardinal for the same purpose – for the technical details. We also use fibrational terminology occasionally when we feel it adds insight, but those not familiar with fibrations can simply ignore such comments.

2 Induction Recursion

The original presentation of induction recursion given by Dybjer [7] was as a schema. Dybjer and Setzer [8] further developed the theory to internalize the concept of an inductive-recursive definition. They developed a finite axiomatization of the theory through the introduction of a special type of codes for inductive-recursive definitions. The following axiomatization which closely follows Dybjer and Setzer [8] presents the syntax of IR as an inductive definition.

Definition 1 (IR codes). *Let $D : \mathsf{type}$. The type of $\mathsf{IR}(D)$ codes has the following constructors:*

$$\frac{d : D}{\iota\, d \,:\, \mathsf{IR}(D)}$$

$$\frac{A : \mathsf{Set} \quad f : A \to \mathsf{IR}(D)}{\sigma_A f \,:\, \mathsf{IR}(D)}$$

$$\frac{A : \mathsf{Set} \quad F : (A \to D) \to \mathsf{IR}(D)}{\delta_A F \,:\, \mathsf{IR}(D)}$$

This is the syntax of induction recursion – it is quite remarkable in our opinion that this most powerful of theories of data types can be presented in such a

simple fashion. These rules have been written in natural-deduction style and we may use the ambient type theory to define, for example, the function f in the code $\sigma_A f$. An example of an IR code is given in Example 5 – this code represents the universe containing the natural numbers and closed under Σ-types given in Equation (1). We now turn to the semantics of induction recursion: we interpret IR codes as functors, and to this end, we use the standard families construction Fam from category theory. We start recalling the definition of the category $\mathsf{Fam}\,\mathbb{C}$ of families of objects of a category $\mathbb{C}$.

Definition 2. *Given a category $\mathbb{C}$, the category $\mathsf{Fam}\,\mathbb{C}$ has objects pairs (X, P) where X is a set and $P : X \to \mathbb{C}$ is a functor which we can think as an X-indexed family of objects of $\mathbb{C}$. A morphism from (X, P) to (Y, Q) is a pair (h, k) where $h : X \to Y$ is a function, and $k : P \dot{\to} Q \circ h$ is a natural transformation.*

Of course, the naturality condition in the definition of a morphism of families is vacuous as the domains of the functors in question are discrete.

Remark 3. For any category $\mathbb{C}$, the category $\mathsf{Fam}\,\mathbb{C}$ always has rich structure:

- $\mathsf{Fam}\,\mathbb{C}$ is fibred over Set (see e.g. Jacobs [16]). We omit here the definitions, but recall the standard splitting cleavage of the fibration $\pi : \mathsf{Fam}\,\mathbb{C} \to \mathsf{Set}$ which is relevant later: a morphism $(h, k) : (X, P) \to (Y, Q)$ is a split cartesian morphism if k is a family of identity morphisms, i.e. if $Q = P \circ h$.
- $\mathsf{Fam}\,\mathbb{C}$ is the free set indexed coproduct completion of $\mathbb{C}$; that is $\mathsf{Fam}\,\mathbb{C}$ has all set indexed coproducts and there is an embedding $\mathbb{C} \to \mathsf{Fam}\,\mathbb{C}$ universal among functors $F : \mathbb{C} \to \mathbb{D}$ where $\mathbb{D}$ is a category with set indexed coproducts. Given an A-indexed collection of objects $(X_a, P_a)_{a:A}$ in $\mathsf{Fam}\,\mathbb{C}$, its A-indexed coproduct is the family $(\sum_{a:A} X_a, [P_a]_{a:A})$.
- $\mathsf{Fam}\,\mathbb{C}$ is cocomplete if and only if $\mathbb{C}$ has all small connected colimits (Carboni and Johnstone [6, dual of Prop. 2.1]).
- Fam is a functor $\mathsf{CAT} \to \mathsf{CAT}$; given $F : \mathbb{C} \to \mathbb{D}$, we get a functor $\mathsf{Fam}(F) : \mathsf{Fam}\,\mathbb{C} \to \mathsf{Fam}\,\mathbb{D}$ by composition: $\mathsf{Fam}(F)(X, P) = (X, F \circ P)$. Here CAT is the category of large categories.

When $\mathbb{C}$ is a discrete category every morphism between families (X, P) and (Y, Q) consists only of functions $h : X \to Y$ such that $P\,x = Q\,(h\,x)$ for all x in X. From a fibrational perspective, this amounts to the restriction to the split cartesian fragment $\mathsf{Fam}\,|\mathbb{C}|$ of the fibration $\pi : \mathsf{Fam}\,\mathbb{C} \to \mathsf{Set}$, for $\mathbb{C}$ an arbitrary category. This observation is crucial for the interpretation of IR codes as functors. Indeed, given a type D, which we think as the discrete (possibly large) set of its terms, we interpret IR codes as functors $\mathsf{Fam}\,D \to \mathsf{Fam}\,D$.

Theorem 4 (IR functors). *Let $D : \mathsf{type}$. Every code $\gamma : \mathsf{IR}(D)$ induces a functor*

$$[\![\gamma]\!] : \mathsf{Fam}\,D \to \mathsf{Fam}\,D$$

Proof. We define $[\![\gamma]\!] : \mathsf{Fam}\, D \to \mathsf{Fam}\, D$ by induction on the structure of the code. We first give the action on objects:

$$\begin{aligned}
[\![\iota\, c]\!](X, P) &= (1, \lambda_.\, c) \\
[\![\sigma_A\, f]\!](X, P) &= \sum_{a\,:A} [\![f\, a]\!](X, P) \\
[\![\delta_A\, F]\!](X, P) &= \sum_{g\,:A \to X} [\![F\, (P \circ g)]\!](X, P)
\end{aligned}$$

We now give the action on morphisms. Let $(h, \mathsf{id}) : (X, P) \to (Y, Q)$ be a morphism in $\mathsf{Fam}\, D$, i.e. $h : X \to Y$ and $Q \circ h = P$.

$$\begin{aligned}
[\![\iota\, c]\!](h, \mathsf{id}) &= (\mathsf{id}_1, \mathsf{id}) \\
[\![\sigma_A\, f]\!](h, \mathsf{id}) &= [\mathsf{in}_a \circ [\![f\, a]\!](h, \mathsf{id})]_{a\,:A} \\
[\![\delta_A\, F]\!](h, \mathsf{id}) &= [\mathsf{in}_{h \circ g} \circ [\![F(Q \circ h \circ g)]\!](h, \mathsf{id})]_{g\,:A \to X}
\end{aligned}$$

Here, the last line type checks since $Q \circ h = P$, hence $Q \circ h \circ g = P \circ g$ and we can apply the induction hypothesis. □

Note how the interpretation of both σ and δ codes makes essential use of coproducts of families as defined in Remark 3. In particular, the interpretation of a code $\delta_B F$ uses as index set of the coproduct the function space $(B \to X)$, which is a set since both B and X are.

Ghani et al. [14] introduces morphisms between IR codes. This makes $\mathsf{IR}(D)$ into a category, and the decoding $[\![-]\!] : \mathsf{IR}(D) \to [\mathsf{Fam}\, D, \mathsf{Fam}\, D]$ can be shown to be a full and faithful functor. We will draw inspiration from this in Section 3 when we generalise the semantics to endofunctors on $\mathsf{Fam}\, \mathbb{C}$ for possibly non-discrete categories $\mathbb{C}$.

We call a data type *inductive-recursive* if it is the initial algebra of a functor induced from an IR code. Let us look at some examples.

Example 5 (A universe closed under dependent sums). In the introduction, we introduced a universe in Equation (1), containing the natural numbers and closed under Σ-types, and claimed that this universe can be defined via an inductive-recursive definition. Indeed, one can easily write down a code $\gamma_{\mathbb{N},\Sigma} : \mathsf{IR}(\mathsf{Set})$ for a functor that will have such a universe as its initial algebra:

$$\gamma_{\mathbb{N},\Sigma} \coloneqq \iota\, \mathbb{N} +_{\mathsf{IR}} \delta_1(X \mapsto \delta_{X*}(Y \mapsto \iota\, \Sigma(X*)\, Y)) \;:\; \mathsf{IR}(\mathsf{Set})$$

Here we have used $\gamma +_{\mathsf{IR}} \gamma' \coloneqq \sigma_2\, (0 \mapsto \gamma; 1 \mapsto \gamma')$ to encode a binary coproduct as a 2-indexed coproduct. Also, in the above, note that $X : 1 \to \mathsf{Set}$ and so $X*$ is simply the application of X to the canonical element of 1. If we decode $\gamma_{\mathbb{N},\Sigma}$, we get a functor which satisfies

$$\begin{aligned}
[\![\gamma_{\mathbb{N},\Sigma}]\!](U, T) &\cong (1, \lambda_.\, \mathbb{N}) + (\Sigma u{:}U\,.\, T(u) \to U, \lambda(u, f).\, \Sigma\, x{:}T(u)\,.\, T(f(x))) \\
&= (1 + \Sigma u{:}U\,.\, T(u) \to U, \mathsf{inl}_ \mapsto \mathbb{N}; \mathsf{inr}(u, f) \mapsto \Sigma\, x{:}T(u)\,.\, T(f(x)))
\end{aligned}$$

so that the initial algebra (U, T) of $[\![\gamma_{\mathbb{N},\Sigma}]\!]$, which satisfies $(U, T) \cong [\![\gamma_{\mathbb{N},\Sigma}]\!](U, T)$ by Lambek's Lemma, indeed satisfies Equation (1).

Example 6 (A universe closed under dependent function spaces). In the same way, we can easily write a down a code for a universe closed under Π-types:

$$\gamma_{\mathbb{N},\Pi} \coloneqq \iota\, \mathbb{N} +_{\mathsf{IR}} \delta_1(X \mapsto \delta_{X*}(Y \mapsto \iota\, \Pi(X*)\, Y)) : \mathsf{IR}(\mathsf{Set})$$

Even though this looks extremely similar to the code in the previous example, we will see in the next section that there is a big semantic difference between them.

3 Positive Inductive-Recursive Definitions

Theorem 4 tells us that IR codes can be interpreted as functors on families built over a discrete category. What happens if we try to interpret IR codes on the category $\mathsf{Fam}\,\mathbb{C}$, and not just on the subcategory $\mathsf{Fam}\,|\mathbb{C}|$? The problem is that if we allow for more general morphisms, we can not prove functoriality of the semantics of a δ code as it stands anymore: it is essential to have an actual equality on the second component of a morphism in $\mathsf{Fam}\,\mathbb{C}$ in order to have a sound semantics (see Example 10 below).

In this section we propose a new axiomatization which enables us to solve this problem. This new theory, which we dub *positive inductive-recursive definitions*, abbreviated IR^+, represents a generalization of IR which allows the interpretation of codes as functors defined on $\mathsf{Fam}\,\mathbb{C}$.

3.1 Syntax and Semantics of $\mathsf{IR}^+(\mathbb{C})$

The crucial insight which guides us when introducing the syntax of IR^+ is to deploy proper functors in the introduction rule of a δ code. This enables us to remove the restriction on morphisms within inductive recursive definitions; indeed, if we know that $F : (A \to \mathbb{C}) \to \mathsf{IR}^+(\mathbb{C})$ is a functor, and not just a function, we do not have to rely on the equality $P \circ g = Q \circ h \circ g$ between objects in $\mathbb{C}^A$, but we can use the second component of a morphism (h, k) in $\mathsf{Fam}\,\mathbb{C}$ to get a map $P \circ g \to Q \circ h \circ g$; then we can use the fact that F is a functor to get a morphism between codes $F(P \circ g) \to F(Q \circ h \circ g)$.

But, now we have to roll up our sleeves. For $F : (A \to \mathbb{C}) \to \mathsf{IR}^+(\mathbb{C})$ to be a functor, we need both $A \to \mathbb{C}$ and $\mathsf{IR}^+(\mathbb{C})$ to be categories. While it is clear how to make $A \to \mathbb{C}$ a category, turning $\mathsf{IR}^+(\mathbb{C})$ into a category entails defining both codes and morphisms between codes simultaneously, in an inductive-inductive fashion [21]. We give an axiomatic presentation of IR^+ analogously to the one given in Section 2 for the syntax of IR; however we now have mutual introduction rules to build both the type of $\mathsf{IR}^+(\mathbb{C})$ codes and the type of $\mathsf{IR}^+(\mathbb{C})$ morphisms, for $\mathbb{C}$ a given category. The semantics we give then explains how $\mathsf{IR}^+(\mathbb{C})$ codes can be interpreted as functors on $\mathsf{Fam}\,\mathbb{C}$, while $\mathsf{IR}^+(\mathbb{C})$ morphisms between such codes can be interpreted as natural transformations.

Definition 7. *Given a category $\mathbb{C}$ we simultaneously define the type of positive inductive-recursive codes on $\mathbb{C}$, $\mathsf{IR}^+(\mathbb{C}) : \mathsf{type}$ and the type of morphisms between these codes $\mathsf{IR}^+(\mathbb{C})(\_,\_) : \mathsf{IR}^+(\mathbb{C}) \to \mathsf{IR}^+(\mathbb{C}) \to \mathsf{type}$ as follows:*

– $\mathsf{IR}^+(\mathbb{C})$ *codes:*

$$\frac{c : \mathbb{C}}{\iota\, c : \mathsf{IR}^+(\mathbb{C})}$$

$$\frac{A : \mathsf{Set} \qquad f : A \to \mathsf{IR}^+(\mathbb{C})}{\sigma_A f : \mathsf{IR}^+(\mathbb{C})}$$

$$\frac{A : \mathsf{Set} \qquad F : (A \to \mathbb{C}) \to \mathsf{IR}^+(\mathbb{C})}{\delta_A F : \mathsf{IR}^+(\mathbb{C})}$$

– $\mathsf{IR}^+(\mathbb{C})$ *morphisms:*

- *morphisms from ιc:*

$$\frac{f : \mathbb{C}(c, c')}{\Gamma_{\iota,\iota}(f) : \mathsf{IR}^+(\mathbb{C})(\iota\, c, \iota\, c')}$$

$$\frac{a : A \qquad r : \mathsf{IR}^+(\mathbb{C})(\iota\, c, f\, a)}{\Gamma_{\iota,\sigma}(a, r) : \mathsf{IR}^+(\mathbb{C})(\iota\, c, \sigma_A f)}$$

$$\frac{g : A \to \emptyset \qquad r : \mathsf{IR}^+(\mathbb{C})(\iota\, c, F(! \circ g))}{\Gamma_{\iota,\delta}(g, r) : \mathsf{IR}^+(\mathbb{C})(\iota\, c, \delta_A F)}$$

- *morphisms from $\sigma_A f$:*

$$\frac{\gamma, : \mathsf{IR}^+(\mathbb{C}) \qquad r : (a : A) \to \mathsf{IR}^+(\mathbb{C})(f a, \gamma)}{\Gamma_{\sigma,\gamma}(r) : \mathsf{IR}^+(\mathbb{C})(\sigma_A f, \gamma)}$$

- *morphisms from $\delta_A F$*

$$\frac{\rho : \mathsf{Nat}(F, \kappa_{\iota c})}{\Gamma_{\delta,\iota}(\rho) : \mathsf{IR}^+(\mathbb{C})(\delta_A F, \iota\, c)}$$

$$\frac{b : B \qquad \rho : \mathsf{Nat}(F, \kappa_{f\, b})}{\Gamma_{\delta,\sigma}(b, \rho) : \mathsf{IR}^+(\mathbb{C})(\delta_A F, \sigma_A f)}$$

$$\frac{g : B \to A \qquad \rho : \mathsf{Nat}(F, G(- \circ g))}{\Gamma_{\delta,\delta}(g, \rho) : \mathsf{IR}^+(\mathbb{C})(\delta_A F, \delta_B G)}$$

In the last three clauses we have indicated with $\kappa_\gamma : \mathbb{C}^A \to \mathsf{IR}^+(\mathbb{C})$ the constant functor with value γ.

We now explain how each code $\gamma : \mathsf{IR}^+(\mathbb{C})$ is interpreted as an endofunctor

$$[\![\gamma]\!] : \mathsf{Fam}\,\mathbb{C} \to \mathsf{Fam}\,\mathbb{C}$$

A functor which is isomorphic to a functor induced by an IR^+ code is called an IR^+ functor. The semantics of IR^+ closely follows the one given in Section 2; as before we make essential use of coproducts in $\mathsf{Fam}\,\mathbb{C}$. Having said that, the crucial feature which separates the semantics of IR^+ from the semantics of IR is the following: when explaining the semantics of IR we can first interpret IR codes as functors and only later we define morphisms between codes which are interpreted as natural transformations between the corresponding functors. In IR^+ the type of of codes and the type of morphisms between codes are simultaneously defined in an inductive-inductive way, and therefore they are also decoded simultaneously as functors and natural transformations respectively. This is exactly what the elimination principle for an inductive-inductive definition gives.

In the following theorem, note that there is no restriction on the category $\mathbb{C}$ – all structure that we need comes for free from the families construction Fam.

Theorem 8 (IR^+ functors). *Let $\mathbb{C}$ be an arbitrary category.*

(i) Every code $\gamma : \mathsf{IR}^+(\mathbb{C})$ induces a functor $[\![\gamma]\!] : \mathsf{Fam}\,\mathbb{C} \to \mathsf{Fam}\,\mathbb{C}$.
(ii) Every morphism $r : \mathsf{IR}^+(\mathbb{C})(\gamma, \gamma')$ for codes $\gamma, \gamma' : \mathsf{IR}^+(\mathbb{C})$ gives rise to a natural transformation $[\![r]\!] : [\![\gamma]\!] \dot{\longrightarrow} [\![\gamma']\!]$.

Proof. While the action on objects is the same for both IR^+ and IR functors, the action on morphisms is different when interpreting a code of type $\delta_A F$: in the semantics of IR^+ we exploit the fact that $F : (A \to \mathbb{C}) \to \mathsf{IR}^+(\mathbb{C})$ is now a functor by using its action on morphism (which we, for the sake of clarity, indicate with $F_\to$). We give the action of IR^+ functors on morphisms only, and refer to the semantics given in Theorem 4 for the action on objects of $\mathsf{Fam}\,\mathbb{C}$.

The action on morphisms is given as follows. Let $(h, k) : (X, P) \to (Y, Q)$ in $\mathsf{Fam}\,\mathbb{C}$. We define $[\![\gamma]\!](h, k) : [\![\gamma]\!](X, P) \to [\![\gamma]\!](Y, Q)$ by recursion on γ:

$$\begin{aligned}
[\![\iota\, c]\!](h, k) &= (\mathsf{id}_1, \mathsf{id}_c) \\
[\![\sigma_A f]\!](h, k) &= [\mathsf{in}_a \circ [\![f\, a]\!](h, k)]_{a:A} \\
[\![\delta_A F]\!](h, k) &= [\mathsf{in}_{h \circ g} \circ [\![F(Q \circ h \circ g)]\!](h, k) \circ [\![F_\to(g^*(k))]\!]_{(X,P)}]_{g:A \to X}
\end{aligned}$$

In the last clause $g^*(k) : P \circ g \dot{\longrightarrow} Q \circ h \circ g$ is the natural transformation with component $g^*(k)_a = k_{g\,a} : P(g\,a) \to Q(k(g\,a))$; note that such a natural transformation is nothing but the vertical morphism above A obtained by reindexing (id_X, k) along g in the families fibration $\pi : \mathsf{Fam}\,\mathbb{C} \to \mathsf{Set}$.

We now explain how a IR^+ morphism $r : \gamma \to \gamma'$ is interpreted as natural transformation $[\![r]\!] : [\![\gamma]\!] \dot{\longrightarrow} [\![\gamma']\!]$ between IR^+ functors by specifying the component $[\![r]\!]_{(X,P)}$ at $(X, P) : \mathsf{Fam}\,\mathbb{C}$. Naturality of these transformations can be proved by a routine diagram chasing.

$$
\begin{aligned}
[\![\Gamma_{\iota,\iota}(f)]\!]_{(X,P)} &= (\mathsf{id}_1, f) \\
[\![\Gamma_{\iota,\sigma}(a,r)]\!]_{(X,P)} &= \mathsf{in}_a \circ [\![r]\!]_{(X,P)} \\
[\![\Gamma_{\iota,\delta}(g,r)]\!]_{(X,P)} &= \mathsf{in}_{!_X \circ g} \circ [\![r]\!]_{(X,P)} \\
[\![\Gamma_{\sigma,\gamma}(r)]\!]_{(X,P)} &= [[\![r\ a]\!]_{(X,P)}]_{a:A} \\
[\![\Gamma_{\delta,\iota}(\rho)]\!]_{(X,P)} &= [[\![\rho_{P \circ g}]\!]_{(X,P)}]_{g:A \to X} \\
[\![\Gamma_{\delta,\sigma}(b,\rho)]\!]_{(X,P)} &= \mathsf{in}_b \circ [[\![\rho_{P \circ g}]\!]_{(X,P)}]_{g:A \to X} \\
[\![\Gamma_{\delta,\delta}(f,\rho)]\!]_{(X,P)} &= [\mathsf{in}_{g \circ f} \circ [\![\rho_{P \circ g}]\!]_{(X,P)}]_{g:A \to X}
\end{aligned}
$$

□

Example 9 (A universe closed under dependent sums in $\mathsf{Fam}\,\mathsf{Set}^{\mathsf{op}}$*).* In Example 5, we defined an ordinary IR code $\gamma_{\mathbb{N},\Sigma} : \mathsf{IR}(\mathsf{Set})$ for a universe closed under sigma types. We can extend this code to an IR^+ code

$$\gamma_{\mathbb{N},\Sigma} = \iota\,\mathbb{N} +_{\mathsf{IR}} \delta_1(X \mapsto \delta_{X*}(Y \mapsto \iota\,\Sigma(X*)\,Y)) : \mathsf{IR}^+(\mathsf{Set}^{\mathsf{op}})$$

where now $G \coloneqq Y \mapsto \iota\,\Sigma(X*)\,Y$ and $F \coloneqq X \mapsto \delta_{X*}\,G$ needs to be functors. Given $f : Y \to Y'$ in $X \to \mathsf{Set}^{\mathsf{op}}$, i.e. $f_x : Y(x) \to Y'(x)$ in $\mathsf{Set}^{\mathsf{op}}$, we have $\Sigma x : (X*).f_x : \Sigma(X*)\,Y \to \iota\,\Sigma(X*)\,Y'$ in $\mathsf{Set}^{\mathsf{op}}$ so that we can define

$$G(f) : \iota\,\Sigma(X*)\,Y \to \iota\,\Sigma(X*)\,Y'$$

by $G(f) = \Gamma_{\iota,\iota}(\Sigma x : (X*).f_x)$.

We also need F to be a functor. Given $f : X \to X'$ in $1 \to \mathsf{Set}^{\mathsf{op}}$, we need to define $F(f) : \delta_{X*}\,G \to \delta_{X'*}\,G$. According to Definition 7, it is enough to give $f_* : X'* \to X*$ and $[\mathsf{in}_{f_* x}]_{x:X'*}$, a natural transformation from G to $G \circ f_*^*$. Notice that working in $\mathsf{Set}^{\mathsf{op}}$ made sure that f_* was going in the right direction.

Example 10 (A universe closed under dependent function spaces in $\mathsf{Fam}\,\mathsf{Set}^{\cong}$*).* In Example 6, we saw how we could use induction-recursion to define a universe closed under Π-types in $\mathsf{Fam}\,|\mathsf{Set}|$, using the following code:

$$\gamma_{\mathbb{N},\Pi} = \iota\,\mathbb{N} +_{\mathsf{IR}} \delta_1(X \mapsto \delta_{X*}(Y \mapsto \iota\,\Pi(X*)\,Y)) : \mathsf{IR}(\mathsf{Set})$$

If we try to extend this to an IR^+ code in $\mathsf{Fam}\,\mathsf{Set}$ or $\mathsf{Fam}\,\mathsf{Set}^{\mathsf{op}}$, we run into problems. Basically, given a morphism $f : X' \to X$, we need to construct a morphism $\Pi\,X'\,(Y \circ f) \to \Pi\,X\,Y$, which of course is impossible if e.g. $X' = 0$, $X = 1$, and $Y* = 0$.

Hence the inherent contravariance in the Π-type means that $\gamma_{\mathbb{N},\Pi}$ doesn't extend to a $\mathsf{IR}^+(\mathsf{Set})$ or $\mathsf{IR}^+(\mathsf{Set}^{\mathsf{op}})$ code. However, if we move to the groupoid $\mathsf{Set}^{\cong}$, which is the subcategory of Set with only isomorphisms as morphisms, we do get an $\mathsf{IR}^+(\mathsf{Set}^{\cong})$ code describing the universe in question, which is still living in a category beyond the strict category $\mathsf{Fam}\,|\mathsf{Set}|$. It would be interesting to understand the relevance of positive induction recursion to homotopy type theory.

4 Application: A Concrete Representation of Nested Types

Nested data types [2] have been used to implement a number of advanced data types in languages which support higher-kinded types, such as the widely-used functional programming language Haskell. Among these data types are those with constraints, such as perfect trees [22]; types with variable binding, such as untyped λ-terms [12]; cyclic data structures [13]; and certain dependent types [20].

A canonical example of a nested data type is $\mathtt{Lam} : \mathsf{Set} \to \mathsf{Set}$ defined in Haskell as follows:

```
data Lam a = Var a | App (Lam a) (Lam a) | Abs (Lam (Maybe a))
```

The type `Lam a` is the type of untyped λ-terms over variables of type `a` up to α-equivalence. Here, the constructor `Abs` models the bound variable in an abstraction of type `Lam a` by the `Nothing` constructor of type `Maybe a`, and any free variable `x` of type `a` in an abstraction of type `Lam a` by the term `Just x` of type `Maybe a`; The key observation about the type `Lam a` is that elements of the type `Lam (Maybe a)` are needed to build elements of `Lam a` so that, in effect, the entire family of types determined by `Lam` has to be constructed simultaneously. Thus, rather than defining a family of inductive types, the type constructor `Lam` defines an *type-indexed inductive family of types*. The kind of recursion captured by nested types is a special case of *non-uniform recursion* [5].

This section asks the question *Are nested data types representable as containers?* There would be benefits of a positive answer in that one could then apply container technology to nested data types, e.g. one could classify the natural transformations between them and operate on them using, for example, the derivative. While the latter has clear practical importance, note that the canonical recursion operator `fold` associated to inductive types is, when analysed for nested data types, a natural transformation.

We give a positive answer to the above question using IR^+. We sketch our overall development as follows:

- we define a grammar Nest for defining nested types and a decoding function $(\!|-|\!) : \mathsf{Nest} \to [\mathsf{Set}, \mathsf{Set}] \to [\mathsf{Set}, \mathsf{Set}]$.
- We show that $(\!|N|\!)$ restricts to an endofunctor $(\!|N|\!)_{\mathsf{Cont}}$ on the category Cont of containers.
- We use IR^+ to define $(\!|N|\!)_{\mathsf{Cont}}$. Hence by the results of this paper, $(\!|N|\!)_{\mathsf{Cont}}$ has an initial algebra $\mu(\!|N|\!)_{\mathsf{Cont}}$. We finish by arguing that $\mu(\!|N|\!) = [\![\mu(\!|N|\!)_{\mathsf{Cont}}]\!]_{\mathsf{Cont}}$ and hence that, indeed, nested types are containers.

A Grammar for Nested Types. We now present a grammar for defining nested data types. It is not the most sophisticated grammar, since our point is not to push the theory of nested data types, but rather to illustrate an application of positive induction-recursion to nested data types. The grammar we use is

$$\mathcal{F} = \mathsf{Id} \mid K\,C \mid \mathcal{F} + \mathcal{F} \mid \mathcal{F} \times \mathcal{F} \mid \mathcal{F} \circledast \mathcal{F}$$

where C is any container. The intention is that Id stands for the identity functor mapping a functor to itself, KC stands for the constant functor mapping any functor to the interpretation of the container C, $+$ stands for the coproduct of functors, $\times$ for the product of functors and $\circledast$ for the pointwise composition of functors. These intentions are formalised by a semantics for the elements of our grammar given as follows

$$\begin{array}{ll}
(\!|-|\!) & : \ \mathsf{Nest} \to [\mathsf{Set},\mathsf{Set}] \to [\mathsf{Set},\mathsf{Set}] \\
(\!|\mathsf{Id}|\!)\ F & = F \\
(\!|K\ C|\!)\ F & = [\![C]\!]_{\mathsf{Cont}} \\
(\!|\mathcal{F} + \mathcal{F}_1|\!)\ F & = (\!|\mathcal{F}|\!)\ F + (\!|\mathcal{F}_1|\!)\ F \\
(\!|\mathcal{F} \times \mathcal{F}_1|\!)\ F & = (\!|\mathcal{F}|\!)\ F \times (\!|\mathcal{F}_1|\!)\ F \\
(\!|\mathcal{F} \circledast \mathcal{F}_1|\!)\ F & = (\!|\mathcal{F}|\!)\ F \circ (\!|\mathcal{F}_1|\!)\ F
\end{array}$$

For example, the functor

$$L\ F\ X \ = \ X \ + \ (FX \times FX) \ + \ F(X+1)$$

whose initial algebra is the type $\mathtt{Lam}$ is of the form $(\!|N_L|\!)$ where

$$N_L = Kid \ + \ (\mathsf{Id} \times \mathsf{Id}) \ + \ \mathsf{Id} \circledast (KM)$$

where id is the container with one shape and one position which represents the identity functor on Set, and M is the container with two positions having one shape above one position and no shapes above the other. M represents the functor on Set mapping X to $X+1$.

Nested Types as Functors on Containers. The next thing on our agenda is to show that every element N of Nest has an interpretation as an operator on containers $(\!|N|\!)_{\mathsf{Cont}} : \mathsf{Cont} \to \mathsf{Cont}$.

$$\begin{array}{ccc}
\mathsf{Cont} & \xrightarrow{[\![-]\!]_{\mathsf{Cont}}} & [\mathsf{Set},\mathsf{Set}] \\
{\scriptstyle (\!|N|\!)_{\mathsf{Cont}}}\downarrow & & \downarrow{\scriptstyle (\!|N|\!)} \\
\mathsf{Cont} & \xrightarrow[[\![-]\!]_{\mathsf{Cont}}]{} & [\mathsf{Set},\mathsf{Set}]
\end{array}$$

This is done easily enough by recursion on N noting that containers are closed under coproduct, product and under composition. Thus, for example, if we define $(\!|N_L|\!)_{\mathsf{Cont}}(S,P)$ to be the container (S_L, P_L) then

$$\begin{array}{ll}
S_L & = 1 + (S \times S) + \Sigma s : S.\ Ps \to 2 \\
P_L\ (\mathsf{in}_1\ *) & = 1 \\
P_L\ (\mathsf{in}_2\ (s,s')) & = Ps + Ps' \\
P_L\ (\mathsf{in}_3\ (s,f)) & = \Sigma p : Ps.\ \mathsf{if}\ fp\ \mathsf{then}\ 1\ \mathsf{else}\ 0
\end{array}$$

Nested Types are Containers. We know that $\mathsf{Cont} = \mathsf{Fam}\,\mathsf{Set}^{\mathsf{op}}$. Now, we want to show that for every code $N : \mathsf{Nest}$, the functor $(\!|N|\!)_{\mathsf{Cont}}$ is a IR^+ functor:

to see this one needs to carefully examine the constructions on families used to build $(\!|N|\!)_{\mathsf{Cont}}$. The only sophisticated construction is the use of Σ-types to model the composition operator used in the definition of nested types and seen in the definition of S_L and P_L. But, as we have seen in Example 9, families closed under Σ are canonical examples of a IR^+ construction. Thus, by the results in Section 6, for every $N : \mathsf{Nest}$, the IR^+ functor $(\!|N|\!)_{\mathsf{Cont}}$ has an initial algebra which is a container (S_N, P_N). Finally, since $[\![-]\!]_{\mathsf{Cont}}$ preserves initial objects and filtered colimits of cartesian morphisms ([1] Propositions 4.5.1 and 4.6.7) and we know from Lemma 14 in Section 6 that the initial algebra chain of an IR^+ functor is made from cartesian morphisms only, we can conclude that $[\![(S_N, P_N)]\!]_{\mathsf{Cont}} = \mu(\!|N|\!)$ showing that all nested types indeed are definable using containers.

5 Comparison to Plain IR

We now investigate the relationship between IR^+ and IR. Note that every type D can be regarded as a discrete category, which we by abuse of notation denote $|D|$. In the other direction, every category $\mathbb{C}$ gives rise to a type $|\mathbb{C}|$ whose elements are the objects of $\mathbb{C}$.

Proposition 11. *There is a function* $\varphi : \mathsf{IR}(D) \to \mathsf{IR}^+(|D|)$ *s.t.*

$$[\![\gamma]\!]_{\mathsf{IR}(D)} \cong [\![\varphi(\gamma)]\!]_{\mathsf{IR}^+(|D|)}$$

Proof. The only interesting case is the δ code. Since $|D|$ is a discrete category, also $A \to |D|$ is discrete. Hence a mapping on objects $(A \to |D|) \to \mathsf{IR}(D)$ can trivially be extended to a functor $(A \to |D|) \to \mathsf{IR}^+(|D|)$. □

This proposition shows that the theory of IR can be embedded in the theory of IR^+. In the next proposition we slightly sharpen this result. We use the functoriality of the Fam construction (Remark 3) to show that forgetting about the extra structure in IR^+ simply gets us back to plain IR.

Proposition 12. *Let* $|-| : \mathsf{CAT} \to \mathsf{SET}$ *be the functor assigning to each category the collection of its objects. There is a function* $\psi : \mathsf{IR}^+\mathbb{C} \to \mathsf{IR}\,|\mathbb{C}|$ *such that*

$$\mathsf{Fam}\,|-| \circ [\![\gamma]\!]_{\mathsf{IR}^+\mathbb{C}} = [\![\psi(\gamma)]\!]_{\mathsf{IR}\,|\mathbb{C}|} \circ \mathsf{Fam}\,|-|$$

for all $\gamma : \mathsf{IR}^+\mathbb{C}$. *Furthermore,* $\psi \circ \varphi = \mathsf{id}$. □

6 Existence of Initial Algebras

We briefly revisit the initial algebra argument used by Dybjer and Setzer [8]. Inspecting their proof, we see that it indeed is possible to adapt it also for the more general setting of positive inductive-recursive definitions by making the appropriate adjustments.

Remember that we call a morphism $(h, k) : (U, T) \to (U, T')$ in $\mathsf{Fam}\,\mathbb{C}$ a *splitting morphism* if $k = \mathsf{id}_T$, i.e. $T' \circ h = T$. We indicate by $\mathsf{Fam}\,|\mathbb{C}|$ the subcategory (subfibration) of $\mathsf{Fam}\,\mathbb{C}$ with the same objects, but with morphisms the splitting ones only.

The proof of existence of initial algebras for IR functors as given by Dybjer and Setzer [8] takes place in the category $\mathsf{Fam}\,|\mathbb{C}|$. The hard work of the proof is split between two lemmas. First Dybjer and Setzer prove that an IR functor $[\![\gamma]\!]$ preserves κ-filtered colimits if κ is an inaccessible cardinal which suitably bounds the size of the index sets in the image of the filtered diagram. Secondly they use the assumption of the existence of a large cardinal, namely a Mahlo cardinal, to prove that such a cardinal bound for the index sets can actually be found. The exact definition of when a cardinal is a Mahlo cardinal will not be important for the current presentation; see Dybjer and Setzer [8] for how this assumption is used. The existence of an initial algebra then follows a standard argument: the initial algebra of a κ-continuous functor can be constructed as the colimit of the initial chain up to κ iterations (see e.g. Adámek et al. [4]).

Inspecting the proofs, we see that they crucially depend on morphisms being splitting in several places. Luckily, the morphisms involved in the corresponding proofs for IR^+ actually are! As is well-known, a weaker condition than κ-continuity is actually sufficient: it is enough that the functor in question preserve the specific colimit of the initial κ-chain. We thus show that the initial chain of a IR^+ functor actually lives in $\mathsf{Fam}\,|\mathbb{C}|$, which will allow us to modify Dybjer and Setzer's proof accordingly.

Lemma 13. *For every code $\gamma : \mathsf{IR}^+\,\mathbb{C}$ the induced functor $[\![\gamma]\!] : \mathsf{Fam}\,\mathbb{C} \to \mathsf{Fam}\,\mathbb{C}$ preserves splitting morphisms, i.e. if (f, g) is splitting, then so is $[\![\gamma]\!](f, g)$.*

Proof. By induction on the structure of the code. The interesting case is $\gamma = \delta_A F$. Let $(h, id) : (X, P \circ h) \to (Y, P)$ be a splitting morphism. We have

$$\begin{aligned}[\![\delta_A F]\!](h, \mathsf{id}) &= [in_{h \circ g} \circ [\![F(P \circ h \circ g)]\!](h, \mathsf{id}) \circ [\![F_{\to}(g^*(\mathsf{id}))]\!]_{(X,P)}]_{g : A \to X} \\ &= [in_{h \circ g} \circ [\![F(P \circ h \circ g)]\!](h, \mathsf{id})]_{g : A \to X}\end{aligned}$$

where $[\![F(g^*\mathsf{id})]\!]_{(X,P)} = \mathsf{id}$ since both g^*, F and $[\![\_]\!]$ are functors. By the induction hypothesis, each $[\![F(P \circ h \circ g)]\!](h, \mathsf{id})$ is splitting. Furthermore injections are splitting in $\mathsf{Fam}\,\mathbb{C}$. Since composition of splitting morphisms is still splitting and the cotuple of splitting morphisms is also splitting in $\mathsf{Fam}\,\mathbb{C}$ we conclude that $[\![\delta_A\, F]\!](h, \mathsf{id})$ is a splitting morphism. □

Lemma 14. *For each $\gamma : \mathsf{IR}^+\,\mathbb{C}$, the initial chain*

$$0 \to [\![\gamma]\!](0) \to [\![\gamma]\!]^2(0) \to \ldots$$

consists of splitting morphisms only.

Proof. Recall that the connecting morphisms $\omega_{j,k} : [\![\gamma]\!]^j(0) \to [\![\gamma]\!]^k(0)$ are uniquely determined as follows:

- $\omega_{0,1} = !_{[\![\gamma]\!](0)}$ is unique.
- $\omega_{j+1,k+1}$ is $[\![\gamma]\!](\omega_{j,k}) : [\![\gamma]\!]([\![\gamma]\!]^j(0)) \to [\![\gamma]\!]([\![\gamma]\!]^k(0))$.
- $\omega_{j,k}$ is the colimit cocone for j a limit ordinal.

We prove the statement by induction on j. It is certainly true that $!_{[\![\gamma]\!](0)} : (0, !) \to [\![\gamma]\!](0)$ is an identity at each component – there are none. Thus $\omega_{0,1}$ is a splitting morphism. At successor stages, we can directly apply Lemma 13 and the induction hypothesis. Finally, at limit stages, we use the fact that the colimit lives in $\mathsf{Fam}\,|\mathbb{C}|$ and hence coincides with the colimit in that category on splitting morphisms, so that the colimit cocone is splitting. □

Inspecting Dybjer and Setzer's original proof, we see that it now goes through also for IR^+ if we insert appeals to Lemma 14 where necessary. To finish the proof, we also need to ensure that $\mathsf{Fam}\,\mathbb{C}$ has κ-filtered colimits; this is automatically true if $\mathbb{C}$ has all small connected colimits (compare Remark 3), since $\mathsf{Fam}\,\mathbb{C}$ then is cocomplete. Note that discrete categories have all small connected colimits for trivial reasons.

Theorem 15. *Assume that a Mahlo cardinal exists in the meta-theory. If $\mathbb{C}$ has connected colimits, then every functor $[\![\gamma]\!]$ for $\gamma : \mathsf{IR}^+\,\mathbb{C}$ has an initial algebra.* □

7 Conclusion

In this paper we have introduced the theory IR^+ of positive inductive-recursive definitions as a generalization of Dybjer and Setzer's theory IR of inductive-recursive definitions [8,9,10], different from the fibrational generalization explored in Ghani et al. [15]: by modifying both syntax and semantics of IR we have been able to broaden the semantics to all of $\mathsf{Fam}\,\mathbb{C}$ and not just $\mathsf{Fam}\,|\mathbb{C}|$. The theory of IR^+, with IR as a subtheory, paves the way to the analysis of more sophisticated data types which allow not only for the simultaneous definition of an inductive type X and of a recursive function $f : X \to D$, but also takes the intrinsic structure between objects in the target type D into account. This is the case for example when D is a setoid, the category Set or $\mathsf{Set}^{\mathsf{op}}$, a groupoid or, even more generally, an arbitrary category $\mathbb{C}$.

In future work we aim to explore the theory of IR^+ from a fibrational perspective: this will allow us to reconcile the theory of IR^+ with the analysis of IR as given in Ghani et al [15]. In particular this will amount to characterising the semantics of δ codes as left Kan extensions. Another interesting direction of research is to investigate to which extent the rich structure of the families construction Fam will help shed light on the analysis of IR^+ types: in particular to exploit the monadic structure of Fam and then to investigate the relationship between the theory of IR^+ and the theory of familial 2-functors introduced by Weber [17].

References

1. Abbott, M.: Category of Containers. Ph.D. thesis, University of Leicester (2003)
2. Abel, A., Matthes, R., Uustalu, T.: Iteration and coiteration schemes for higher-order and nested datatypes. Theoretical Computer Science 333(1-2), 3–66 (2005)
3. Aczel, P.: Frege structures and the notions of proposition, truth and set. In: Barwise, J., Keisler, H.J., Kunen, K. (eds.) The Kleene Symposium. Studies in Logic and the Foundations of Mathematics, vol. 101, pp. 31–59. Elsevier (1980)
4. Adámek, J., Milius, S., Moss, L.: Initial algebras and terminal coalgebras: A survey (June 29, 2010), draft
5. Blampied, P.: Structured Recursion for Non-uniform Data-types. Ph.D. thesis, University of Nottingham (2000)
6. Carboni, A., Johnstone, P.: Connected limits, familial representability and Artin glueing. Mathematical Structures in Computer Science 5(4), 441–459 (1995)
7. Dybjer, P.: A general formulation of simultaneous inductive-recursive definitions in type theory. Journal of Symbolic Logic 65(2), 525–549 (2000)
8. Dybjer, P., Setzer, A.: A finite axiomatization of inductive-recursive definitions. In: Girard, J.-Y. (ed.) TLCA 1999. LNCS, vol. 1581, pp. 129–146. Springer, Heidelberg (1999)
9. Dybjer, P., Setzer, A.: Induction–recursion and initial algebras. Annals of Pure and Applied Logic 124(1-3), 1–47 (2003)
10. Dybjer, P., Setzer, A.: Indexed induction–recursion. Journal of Logic and Algebraic Programming 66(1), 1–49 (2006)
11. Ek, L., Holmström, O., Andjelkovic, S.: Formalizing Arne Andersson trees and Left-leaning Red-Black trees in Agda. Bachelor thesis, Chalmers University of Technology (2009)
12. Fiore, M., Plotkin, G., Turi, D.: Abstract syntax and variable binding. In: Proc. Logic in Computer Science, pp. 193–202 (1999)
13. Ghani, N., Hamana, M., Uustalu, T., Vene, V.: Representing cyclic structures as nested types (2006), presented at Trends in Functional Programming
14. Hancock, P., McBride, C., Ghani, N., Malatesta, L., Altenkirch, T.: Small induction recursion. In: Hasegawa, M. (ed.) TLCA 2013. LNCS, vol. 7941, pp. 156–172. Springer, Heidelberg (2013)
15. Ghani, N., Malatesta, L., Nordvall Forsberg, F., Setzer, A.: Fibred data types. In: LICS 2013 (2013)
16. Jacobs, B.: Categorical Logic and Type Theory. Studies in Logic and the Foundations of Mathematics, vol. 141. North Holland, Elsevier (1999)
17. Mark, W.: Familial 2-functors and parametric right adjoints. Theory and Applications of Category Theory 18(22), 665–732 (2007)
18. Martin-Löf, P.: An intuitionistic theory of types (1972), published in Twenty-Five Years of Constructive Type Theory
19. Martin-Löf, P.: Intuitionistic type theory. Bibliopolis Naples (1984)
20. McBride, C., McKinna, J.: The view from the left. Journal of Functional Programming 14(1), 69–111 (2004)
21. Nordvall Forsberg, F., Setzer, A.: A finite axiomatisation of inductive-inductive definitions. In: Berger, U., Hannes, D., Schuster, P., Seisenberger, M. (eds.) Logic, Construction, Computation, Ontos Mathematical Logic, vol. 3, pp. 259–287. Ontos Verlag (2012)
22. Ralf, H.: Functional pearl: Perfect trees and bit-reversal permutation. Journal of Functional Programming 10(3), 305–317 (2000)

Coalgebraic Up-to Techniques

Damien Pous*

CNRS, LIP, ENS Lyon, France
damien.pous@ens-lyon.fr

1 The Concrete Case of Finite Automata

A simple algorithm for checking language equivalence of finite automata consists in trying to compute a *bisimulation* that relates them. This is possible because language equivalence can be characterised coinductively, as the largest bisimulation.

More precisely, consider an automaton $\langle S, t, o\rangle$, where S is a (finite) set of states, $t : S \to \mathcal{P}(S)^A$ is a non-deterministic transition function, and $o : S \to 2$ is the characteristic function of the set of accepting states. Such an automation gives rise to a determinised automaton $\langle \mathcal{P}(S), t^\sharp, o^\sharp\rangle$, where $t^\sharp : \mathcal{P}(S) \to \mathcal{P}(S)^A$ and $o^\sharp : \mathcal{P}(S) \to 2$ are the natural extensions of t and o to sets. A *bisimulation* is a relation R between sets of states such that for all sets of states X, Y, $X\ R\ Y$ entails:

1. $o^\sharp(X) = o^\sharp(Y)$, and
2. for all letter a, $t^\sharp_a(X)\ R\ t^\sharp_a(Y)$.

The coinductive characterisation is the following one: *two sets of states recognise the same language if and only if they are related by some bisimulation.*

Taking inspiration from concurrency theory [4,5], one can improve this proof technique by weakening the second item in the definition of bisimulation: given a function f on binary relations, a *bisimulation up to f* is a relation R between states such that for all sets X, Y, $X\ R\ Y$ entails:

1. $o^\sharp(X) = o^\sharp(Y)$, and
2. for all letter a, $t^\sharp_a(X)\ f(R)\ t^\sharp_a(Y)$.

For well-chosen functions f, bisimulations up to f are contained in a bisimulation, so that the improvement is sound. So is the function mapping each relation to its equivalence closure. In this particular case, one recover the standard algorithm by Hopcroft and Karp [2]: two sets can be skipped whenever they can already be related by a sequence of pairwise related states.

One can actually do more, by considering the function c mapping each relation to its congruence closure: the smallest equivalence relation which contains the argument, and which is compatible w.r.t. set union:

* Work partially funded by the PiCoq and PACE projects, ANR-10-BLAN-0305 and ANR-12IS02001.

R. Heckel and S. Milius (Eds.): CALCO 2013, LNCS 8089, pp. 34–35, 2013.

$$\frac{}{X\ c(R)\ X} \qquad \frac{Y\ c(R)\ X}{X\ c(R)\ Y} \qquad \frac{X\ c(R)\ Y \qquad Y\ c(R)\ Z}{X\ c(R)\ Z}$$

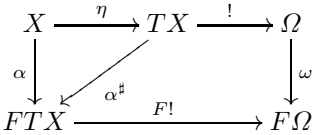
$$\frac{X\ R\ Y}{X\ c(R)\ Y} \qquad \frac{X_1\ c(R)\ Y_1 \qquad X_2\ c(R)\ Y_2}{X_1 \cup X_2\ c(R)\ Y_1 \cup Y_2}\ .$$

This is how we obtained HKC [1], an algorithm that can be exponentially faster than Hopcroft and Karp's algorithm or more recent antichain algorithms [7].

2 Generalisation to Coalgebra

The above ideas generalise nicely, using the notion of λ-bialgebras [3].

Let T be a monad, F an endofunctor, and λ a distributive law $TF \Rightarrow FT$, a λ-*bialgebra* is a triple $\langle X, \alpha, \beta\rangle$, where $\langle X, \alpha\rangle$ is a F-coalgebra, $\langle X, \beta\rangle$ a T-algebra, and $\alpha \circ \beta = F\beta \circ \lambda_X \circ T\alpha$. Given such a λ-bialgebra, FT-algebra generalise non-deterministic automata: take $X \mapsto 2 \times X^A$ for F, and $X \mapsto \mathcal{P}_f X$ for T. Determinisation through the powerset construction can be generalised as follows [6], when the functor F has a final coalgebra $\langle \Omega, \omega\rangle$:



Bisimulations up-to can be expressed in a natural way in such a framework. One can in particular consider bisimulations up to congruence, where the congruence is taken w.r.t. the monad T: the fact that λ is a distributive law ensures that this improvement is always sound.

References

1. Bonchi, F., Pous, D.: Checking NFA equivalence with bisimulations up to congruence. In: POPL, pp. 457–468. ACM (2013)
2. Hopcroft, J.E., Karp, R.M.: A linear algorithm for testing equivalence of finite automata. Technical Report 114, Cornell University (December 1971)
3. Klin, B.: Bialgebras for structural operational semantics: An introduction. TCS 412(38), 5043–5069 (2011)
4. Milner, R.: Communication and Concurrency. Prentice-Hall (1989)
5. Sangiorgi, D.: On the bisimulation proof method. Mathematical Structures in Computer Science 8, 447–479 (1998)
6. Silva, A., Bonchi, F., Bonsangue, M., Rutten, J.: Generalizing the powerset construction, coalgebraically. In: Proc. FSTTCS. LIPIcs, vol. 8, pp. 272–283. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2010)
7. De Wulf, M., Doyen, L., Henzinger, T.A., Raskin, J.-F.: Antichains: A new algorithm for checking universality of finite automata. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 17–30. Springer, Heidelberg (2006)

Exploiting Algebraic Laws to Improve Mechanized Axiomatizations*

Luca Aceto[1], Eugen-Ioan Goriac[1], Anna Ingolfsdottir[1], Mohammad Reza Mousavi[2], and Michel A. Reniers[3]

[1] ICE-TCS, School of Computer Science, Reykjavik University, Menntavegur 1, IS-101, Reykjavik, Iceland
[2] Center for Research on Embedded Systems (CERES), Halmstad University, Sweden
[3] Department of Mechanical Engineering, Eindhoven University of Technology, P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands

Abstract. In the field of structural operational semantics (SOS), there have been several proposals both for syntactic rule formats guaranteeing the validity of algebraic laws, and for algorithms for automatically generating ground-complete axiomatizations. However, there has been no synergy between these two types of results. This paper takes the first steps in marrying these two areas of research in the meta-theory of SOS and shows that taking algebraic laws into account in the mechanical generation of axiomatizations results in simpler axiomatizations. The proposed theory is applied to a paradigmatic example from the literature, showing that, in this case, the generated axiomatization coincides with a classic hand-crafted one.

1 Introduction

Algebraic properties, such as commutativity, associativity and idempotence of binary operators, specify some natural properties of programming and specification constructs. These properties can either be validated using the semantics of the language with respect to a suitable notion of program equivalence, or they can be guaranteed a priori 'by design'. In particular, for languages equipped with a Structural Operational Semantics (SOS) [19], there are two closely related lines of work to achieve this goal: firstly, there is a rich body of syntactic rule formats that can guarantee the validity of certain algebraic properties; see [5,17] for recent surveys. Secondly, there are numerous results regarding the mechanical generation of ground-complete axiomatizations of various behavioral equivalences and preorders for SOS language specifications in certain formats—see, e.g., [1,7,20]. However, these two lines of research have evolved separately and no link has been established between the two types of results so far. In this paper, we take the first steps in marrying these two research areas and in using rule formats for algebraic properties (specifically, for commutativity) to enhance the

* The first three authors have been partially supported by the project 'Meta-theory of Algebraic Process Theories' (nr. 100014021) of the Icelandic Research Fund. Eugen-Ioan Goriac is also funded by the project 'Extending and Axiomatizing Structural Operational Semantics: Theory and Tools' (nr. 1102940061) of the Icelandic Research Fund.

R. Heckel and S. Milius (Eds.): CALCO 2013, LNCS 8089, pp. 36–50, 2013.

process of automatic generation of axiomatizations for strong bisimilarity from GSOS language specifications [10]. In particular, we show that linking these two areas results in axiomatizations that look like hand-crafted ones.

Contribution and Related Work. Many ground-completeness results have been presented in the literature on process calculi. (See, for instance, the survey paper [3] for pointers to the literature.) A common proof strategy for establishing such ground-completeness results is to reduce the problem of axiomatizing the notion of behavioural equivalence under consideration over arbitrary closed terms to that of axiomatizing it over 'synchronization-tree terms'. This approach is also at the heart of the algorithm proposed in [1] for the automatic generation of finite, equational, ground-complete axiomatizations for bisimilarity over language specifications in the GSOS format. A variation on that algorithm for GSOS language specifications with termination has been presented in [7]. In [20], Ulidowski has instead offered algorithms for the automatic generation of finite axiom systems for the testing preorder over de Simone process languages. In Section 4 of this paper, we present a refinement of the algorithm from [1] that uses a rule format guaranteeing commutativity of certain operators to obtain ground-complete axiomatizations of bisimilarity that are closer to the hand-crafted ones than those produced by existing algorithms. (See Section 5, where we apply the algorithm to axiomatize the classic parallel composition operator and compare the generated axiomatization to earlier ones.)

Our rule format for commutativity (presented in Section 3) is a generalization of the rule format for commutativity from [16], which allows operators to have various sets of commutative arguments. Apart from being natural, such a generalization is useful in the automatic generation of ground-complete axiomatizations, as the developments in this study show.

2 Preliminaries

In this section we review, for the sake of completeness, some standard definitions from process theory and the meta-theory of SOS that will be used in the remainder of the paper. We refer the interested reader to [4,17] for further details.

Transition System Specifications in GSOS Format. We let V denote an infinite set of variables with typical members $x, x', x_i, y, y', y_i, \ldots$. A *signature* Σ is a set of function symbols, each with a fixed arity. We call these symbols *operators* and usually represent them by $f, g, \ldots$. An operator with arity zero is called a *constant*. We define the set $\mathbb{T}(\Sigma)$ of *terms* over Σ (sometimes referred to as Σ-terms) as the smallest set satisfying the following constraints: (1) A variable $x \in V$ is a term. (2) If $f \in \Sigma$ has arity n and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term. We use $s, t, t', t_i, u, \ldots$ to range over terms. We write $t_1 \equiv t_2$ if t_1 and t_2 are syntactically equal. The function $vars : \mathbb{T}(\Sigma) \to 2^V$ gives the set of variables appearing in a term. The set $\mathbb{C}(\Sigma)$ is the set of *closed terms*, i.e., the set of all terms t such that $vars(t) = \emptyset$. We use $p, p', p_i, q, r \ldots$ to range over closed terms. A *substitution* σ is a function of type $V \to \mathbb{T}(\Sigma)$. We extend the domain of substitutions to terms homomorphically. If the range of a substitution lies in $\mathbb{C}(\Sigma)$, we say that it is a *closed substitution*.

The GSOS format is a widely studied format of deduction rules in transition system specifications proposed by Bloom, Istrail and Meyer [10]. Transition system specifications whose rules are in the GSOS format enjoy many desirable properties, and several studies in the literature on the meta-theory of SOS have focused on them—see, e.g., the survey [4]. Following [1], in this study we shall also focus on transition system specifications in the GSOS format, which we now proceed to define.

Definition 1 (GSOS Format [10]). *A deduction rule for an operator f of arity n is in the GSOS format if and only if it has the following form:*

$$\frac{\{x_i \xrightarrow{l_{ij}} y_{ij} \mid 1 \leq i \leq n, 1 \leq j \leq m_i\} \cup \{x_i \overset{l_{ik}}{\not\longrightarrow} \mid 1 \leq i \leq n, 1 \leq k \leq n_i\}}{f(\overrightarrow{x}) \xrightarrow{l} C[\overrightarrow{x}, \overrightarrow{y}]}$$

where the x_i's and the y_{ij}'s ($1 \leq i \leq n$ and $1 \leq j \leq m_i$) are all distinct variables, m_i and n_i are natural numbers, $C[\overrightarrow{x}, \overrightarrow{y}]$ is a Σ-term with variables including at most the x_i's and the y_{ij}'s, and the l_{ij}'s and l are labels. If $m_i > 0$, for some i, then we say that the rule tests its i-th argument positively. The above rule is said to be f-defining *and* l-emitting.

A transition system specification (TSS) in the GSOS format *$\mathcal{T}$ is a triple (Σ, L, D) where Σ is a finite signature, L is a finite set of labels, and D is a finite set of deduction rules in the GSOS format. We shall sometimes refer to a TSS in the GSOS format as a* GSOS system.

In addition to the syntactic restrictions on deduction rules, the GSOS format, as presented in [10], requires the signature to include a constant $\mathbf{0}$, a collection of unary operators $a.\_$ ($a \in L$) and a binary operator $\_+\_$. Intuitively, $\mathbf{0}$ represents a process that does not exhibit any behaviour, $s + t$ is the nondeterministic choice between the behaviours of s and t, while $a.t$ is a process that first performs action a and behaves like t afterwards. The standard deduction rules for these operations are given below:

$$\frac{}{a.x_1 \xrightarrow{a} x_1} \qquad \frac{x_1 \xrightarrow{a} x_1'}{x_1 + x_2 \xrightarrow{a} x_1'} \qquad \frac{x_2 \xrightarrow{a} x_2'}{x_1 + x_2 \xrightarrow{a} x_2'}.$$

In the remainder of this paper, following [10], we shall tacitly assume that each TSS in the GSOS format contains these operators with the rules given above. The import of this assumption is that, as is well known, within each TSS in the GSOS format it is possible to express each finite synchronization tree over L. Following [12], the TSS containing the operators $\mathbf{0}$, $a.\_$ ($a \in L$) and $\_+\_$, with the above-given rules, is denoted by BCCSP.

The transition relation associated with a TSS in the GSOS format is the one defined by structural induction over closed terms using the rules. We refer the interested reader to [10] for the precise definition and much more information on GSOS languages.

Definition 2 ([1]). *A GSOS system $\mathcal{T}'$ is a* disjoint extension *of a GSOS system $\mathcal{T}$, denoted by $\mathcal{T} \sqsubseteq \mathcal{T}'$, if the signature and rules of $\mathcal{T}'$ include those of $\mathcal{T}$, and $\mathcal{T}'$ introduces no new rules for operators in the signature of $\mathcal{T}$.*

Bisimilarity and Axiom Systems. The notion of behavioural equivalence that we will use in this paper is the following, classic notion of bisimilarity [15,18].

Definition 3. *Let $\mathcal{T}$ be a GSOS system with signature Σ. A relation $R \subseteq \mathbb{C}(\Sigma) \times \mathbb{C}(\Sigma)$ is a* bisimulation *if and only if R is symmetric and, for all $p_0, p_1, p_0' \in \mathbb{C}(\Sigma)$ and $l \in L$,*

$$(p_0 \; R \; p_1 \wedge p_0 \xrightarrow{l} p_0') \Rightarrow \exists p_1' \in \mathbb{C}(\Sigma).\ (p_1 \xrightarrow{l} p_1' \wedge p_0' \; R \; p_1').$$

Two terms $p_0, p_1 \in \mathbb{C}(\Sigma)$ are called bisimilar*, denoted by $\mathcal{T} \vdash p_0 \leftrightarroweq p_1$ (or simply by $p_0 \leftrightarroweq p_1$ when $\mathcal{T}$ is clear from the context), when there exists a bisimulation R such that $p_0 \; R \; p_1$.*

It is well known that $\leftrightarroweq$ is an equivalence relation over $\mathbb{C}(\Sigma)$. Any equivalence relation $\sim$ over closed terms in a TSS $\mathcal{T}$ is extended to open terms in the standard fashion, i.e., for all $t, u \in \mathbb{T}(\Sigma)$, the equation $t = u$ holds over $\mathcal{T}$ modulo $\sim$ (sometimes abbreviated to $t \sim u$) if, and only if, $\mathcal{T} \vdash \sigma(t) \sim \sigma(u)$ for each closed substitution σ.

Remark 1. If $\mathcal{T}'$ is a disjoint extension of $\mathcal{T}$, then two closed terms over the signature of $\mathcal{T}$ are bisimilar in $\mathcal{T}$ if and only if they are bisimilar in $\mathcal{T}'$.

Proposition 1 ([10]). *$\leftrightarroweq$ is a congruence for any TSS in GSOS format—that is, for all $f \in \Sigma$ and terms $t_1, \ldots, t_n, u_1, \ldots, u_n$, where n is the arity of f, if $t_i \leftrightarroweq u_i$ for each $i \in \{1, \ldots, n\}$ then $f(t_1, \ldots, t_n) \leftrightarroweq f(u_1, \ldots, u_n)$.*

Definition 4 (Axiom System). *An* axiom system *E over a signature Σ is a set of equalities of the form $t = t'$, where $t, t' \in \mathbb{T}(\Sigma)$. An equality $t = t'$, for some $t, t' \in \mathbb{T}(\Sigma)$, is derivable from E, denoted by $E \vdash t = t'$, if and only if it is in the smallest congruence relation over Σ-terms induced by the equalities in E.*

In the context of a fixed TSS $\mathcal{T}$, an axiom system E (over the same signature) is sound *with respect to a congruence relation $\sim$ if and only if for all $t, t' \in \mathbb{T}(\Sigma)$, if $E \vdash t = t'$, then it holds that $\mathcal{T} \vdash t \sim t'$. The axiom system E is* ground complete *if the implication holds in the opposite direction whenever t and t' are closed terms.*

3 Commutativity Format

Commutativity is an essential property specifying that the order of arguments of an operator is immaterial. In the setting of process algebras, commutativity is defined with respect to a notion of behavioural equivalence over terms. In this section, we first present a generalized notion of commutativity that allows n-ary operators to have various sets of commutative arguments and then slightly adapt the commutativity rule format proposed in [16] to the extended setting. Moreover, we give some auxiliary definitions that will be used in the axiomatization procedure proposed in the next section.

In order to motivate the generalized notion of commutativity we present below, consider, by way of example, the ternary operator f defined by the rules below, where a ranges over the collection of action labels L.

$$\frac{x \xrightarrow{a} x'}{f(x,y,z) \xrightarrow{a} f(x',y,z)} \quad \frac{y \xrightarrow{a} y'}{f(x,y,z) \xrightarrow{a} f(x,y',z)}$$

$$\frac{x \xrightarrow{a} x' \;\; z \xrightarrow{a} z'}{f(x,y,z) \xrightarrow{a} f(x',y,z')} \quad \frac{y \xrightarrow{a} y' \;\; z \xrightarrow{a} z'}{f(x,y,z) \xrightarrow{a} f(x,y',z')}.$$

It is not hard to show that the operator f is commutative in its first two arguments modulo bisimilarity, irrespective of the other operators in the TSS under consideration—that is, $f(p,q,r) \leftrightarrow f(q,p,r)$, for all closed terms p,q,r. On the other hand, the third argument does not commute with respect to the other two. For example, we have that $f(a.\mathbf{0},\mathbf{0},\mathbf{0}) \leftrightarrow f(\mathbf{0},\mathbf{0},a.\mathbf{0})$ because $f(a.\mathbf{0},\mathbf{0},\mathbf{0}) \xrightarrow{a} f(\mathbf{0},\mathbf{0},\mathbf{0})$, but $f(\mathbf{0},\mathbf{0},a.\mathbf{0})$ has no outgoing transitions.

The commutativity format presented in [16] can only deal with operators that are commutative for each pair of arguments and, unlike the format that we present below, is therefore unable to detect that f is commutative in its first two arguments.

In what follows, we shall often use $[n]$, $n \geq 0$, to stand for the set $\{1,\ldots,n\}$. Note that $[0]$ is just the empty set.

Definition 5 (Generalized Commutativity). *Given a set I, a family $\prod_I$ of non-empty, pairwise disjoint subsets of I is called a* partition *of I when $\bigcup \prod_I = I$.*

*Let Σ be a signature. Assume that $f \in \Sigma$ is an n-ary operator, $\prod_{[n]}$ is a partition of $[n]$ and $\sim$ is an equivalence relation over $\mathbb{C}(\Sigma)$. The operator f is called $\prod_{[n]}$-*commutative with respect to $\sim$ *when, for each $K \in \prod_{[n]}$ and each two $j,k \in K$ such that $j<k$, the following equation is sound with respect to $\sim$:*

$$f(x_1,\ldots,x_n) = f(x_1,\ldots,x_{j-1},x_k,x_{j+1},\ldots,x_{k-1},x_j,x_{k+1},\ldots,x_n).$$

Note that the traditional notion of commutativity for binary operators can be recovered using Definition 5 in terms of $\{\{1,2\}\}$-commutativity. Moreover, the notion of commutativity for n-ary operators from [16] corresponds to $\{[n]\}$-commutativity. Any n-ary operator is $1_{[n]}$-commutative with respect to any equivalence relation $\sim$, where $1_{[n]} = \{\{1\},\ldots,\{n\}\}$ is the discrete partition of $[n]$.

From this point onward, whenever a signature Σ is provided, we also assume that every function symbol $f \in \Sigma$ of arity n has an associated fixed partition of its set of arguments $[n]$ denoted by $\prod_f$. We denote the indexed set of all these partitions by $\prod^{\Sigma} = \{\prod_f\}_{f\in\Sigma}$.

Definition 6. *Assume $\Sigma_1 \subseteq \Sigma_2$. Let $\prod^{\Sigma_1}$ be a family of partitions. The* extension of $\prod^{\Sigma_1}$ to Σ_2 *is obtained by taking $\prod_f$ to be the discrete partition over $[n]$ for each $f \in \Sigma_2 \setminus \Sigma_1$, where n is the arity of f.*

Our aim is to define a restriction of the GSOS rule format that guarantees the notion of generalized commutativity defined above for any behavioural equivalence that is coarser than bisimilarity. To this end, we begin by extending the notion of commutative congruence introduced in [16] to the context of this generalized notion of commutativity.

Definition 7 (Commutative Congruence). *Consider a signature Σ and a set of partitions $\prod^{\Sigma}$. The commutative congruence relation $\sim_{cc}$ (with respect to $\prod^{\Sigma}$) is the least congruence relation over $\mathbb{T}(\Sigma)$ satisfying the following requirement: for all $f \in \Sigma$ (of arity n), $K \in \prod_f$, $j,k \in K$ with $j<k$, and $t_1,\ldots,t_n \in \mathbb{T}(\Sigma)$, it holds that*

$$f(t_1,\ldots,t_n) \sim_{cc} f(t_1,\ldots,t_{j-1},t_k,t_{j+1},\ldots,t_{k-1},t_j,t_{k+1},\ldots,t_n).$$

We are now ready to present a syntactic restriction on the GSOS format that guarantees commutativity with respect to a set of partitions $\prod^{\Sigma}$ modulo any notion of behavioural equivalence that includes strong bisimilarity. Unlike the format for $\{[n]\}$-commutativity given in [16], the format offered below applies to generalized commutativity, in the sense of Definition 5, and is defined for TSSs whose rules can have negative premises. On the other hand, unlike ours, the format introduced in [16] applies to rules whose positive premises need not have variables as their sources and targets. Extending our format in order to accommodate this kind of premises in deduction rules is straightforward, but is not relevant for the purpose of this paper.

Definition 8 (Comm-GSOS). *A transition system specification over signature Σ is in the comm-GSOS format with respect to a set of partitions $\prod^{\Sigma}$ if it is in the GSOS format and for each f-defining deduction rule $d = \dfrac{H}{f(x_1,\ldots,x_n) \xrightarrow{l} t}$, each $K \in \prod_f$ and for all $j,k \in K$ with $j < k$, there exist a deduction rule $d' = \dfrac{H'}{f(x'_1,\ldots,x'_n) \xrightarrow{l} t'}$ and a bijective mapping $\hbar$ over variables such that*

- *$\hbar(x'_i) = x_i$ for each $i \in [n]$ such that $i \neq j$ and $i \neq k$,*
- *$\hbar(x'_j) = x_k$ and $\hbar(x'_k) = x_j$,*
- *$\hbar(t') \sim_{cc} t$, and*
- *$\hbar(H') = H$.*

Deduction rule d' is called a commutative mirror *of d (with respect to j, k and $\prod^{\Sigma}$).*

The above format requires that, when $f \in \Sigma$, for each f-defining rule and for each pair (j,k) of arguments for which f is supposed to be commutative, as specified by $\prod_f$, there exists a commutative mirror that enables the 'same transitions up to the commutative congruence $\sim_{cc}$ associated with $\prod^{\Sigma}$' when the jth and kth arguments of f are swapped. This is the essence of the proof of the following theorem, which states the correctness of the syntactic comm-GSOS format.

Theorem 1. *If a transition system specification is in the comm-GSOS format with respect to a set of partitions $\prod^{\Sigma}$, then each operator $f \in \Sigma$ is $\prod_f$-commutative with respect to any notion of behavioural equivalence that includes bisimilarity.*

Example 1. Consider the ternary operator f we used earlier to motivate the notion of generalized commutativity. Any transition system specification including the operator f is in the comm-GSOS format with respect to any set of partitions $\prod^{\Sigma}$ such that $\prod_f = \{\{1,2\},\{3\}\}$. Indeed, the a-emitting rules in the first row are one the commutative mirror of the other with respect to $\prod^{\Sigma}$, and so are those in the second row. The constraints in Definition 8 are vacuously satisfied when we take $K = \{3\}$. Therefore, by Theorem 1, we recover the fact that f is commutative in its first two arguments.

Example 2 (Parallel Composition). A frequently occurring commutative operator is parallel composition. It appears in, amongst others, ACP [9], CCS [15], and CSP [14]. Here we discuss parallel composition with communication in the style of ACP [9], of

which the others are special cases. The rules for this operator are listed below. In those rules, a, b, c range over L and $\gamma : L \times L \hookrightarrow L$ is a partial communication function.

$$(\mathrm{p}_1)\ \frac{x \xrightarrow{a} x'}{x \,||\, y \xrightarrow{a} x' \,||\, y} \qquad (\mathrm{p}_2)\ \frac{y \xrightarrow{a} y'}{x \,||\, y \xrightarrow{a} x \,||\, y'} \qquad (\mathrm{p}_3)\ \frac{x \xrightarrow{a} x' \quad y \xrightarrow{b} y'}{x \,||\, y \xrightarrow{c} x' \,||\, y'} \quad \gamma(a,b) = c$$

If the partial communication function γ is commutative, then any GSOS system including the operator $||$ given by the above rules is in the comm-GSOS format with respect to any set of partitions $\prod^{\Sigma}$ such that $\prod_{||} = \{\{1,2\}\}$. Hence it follows from Theorem 1 that $||$ is $\{\{1,2\}\}$-commutative.

4 Mechanized Axiomatization

In this section, we present a technique for the automatic generation of ground-complete axiomatizations of bisimilarity over TSSs in the comm-GSOS format, which is derived from the one introduced in [1]. Our approach improves upon the one in [1] by making use of the rule format for generalized commutativity we introduced in the previous section. Our goal is to generate a disjoint extension of the original TSS and a finite axiom system that is sound and ground complete for bisimilarity over it. This finite axiom system may then also be used for equationally establishing bisimilarity over closed terms from the original TSS. We start by axiomatizing a rather restrictive subset of 'good' operators in Section 4.1. Then we turn 'bad' operators into good ones by means of auxiliary operators. In both of these steps, we exploit commutativity information, where possible, in order to reduce the number of generated axioms, as well as the number of generated auxiliary operators.

4.1 Axiomatizing Good Operators

The approach offered in [1] relies on the fact that the signature includes the operators from BCCSP. (Recall that, in keeping with [10], we assume that these operators are present in any TSS in the GSOS format.) The aim of the axiomatization procedure is then to generate an axiom system that can rewrite any closed term p into a term p' in head normal form such that $p \leftrightarrows p'$. (We call an axiom system with this property *head normalizing*.) Recall that a term t is in *head normal form* if it has the form $a_1.t_1 + \cdots + a_n.t_n$ for some $n \geq 0$, some set of actions $\{a_i \mid i \in [n]\}$ and set of terms $\{t_i \mid i \in [n]\}$. If $n = 0$ then $a_1.t_1 + \cdots + a_n.t_n$ stands for $\mathbf{0}$.

For 'semantically well founded' terms (see [1, Definition 5.1 on page 28]), rewriting into head normal form can be used to prove that each closed term is equal to a closed term over the signature for BCCSP. This leads to a ground-complete axiomatization of bisimilarity, since BCCSP is finitely axiomatized modulo bisimilarity by the axiom system E_{BCCSP} from [13] consisting of the axioms stating that '$+$' is associative, commutative, idempotent and has $\mathbf{0}$ as unit element.

To start with, we focus on the case of closed terms built using only *good* operators, which we now proceed to define.

Definition 9 (Smooth and distinctive operator). *Consider an n-ary operator f.*

1. *A* smooth GSOS deduction rule *is of the form*

$$\frac{\{x_i \xrightarrow{a_i} y_i \mid i \in I\} \cup \{x_i \overset{b_{ij}}{\nrightarrow} \mid i \in J, 1 \leq j \leq n_i\}}{f(x_1, \ldots, x_n) \xrightarrow{c} C[\overrightarrow{x}, \overrightarrow{y}]}$$

where I and J are disjoint subsets of $[n]$ such that $I \cup J = [n]$, and $C[\overrightarrow{x}, \overrightarrow{y}]$ can only include the variables x_i ($i \in [n] \setminus I$) and y_i ($i \in I$).
An operator f of a TSS in the GSOS format is smooth *if all its rules are smooth.*

2. *An n-ary operator f of a TSS in the GSOS format is* distinctive *if it is smooth, each f-defining rule tests the same set of arguments I positively, and for every two distinct f-defining rules there is some argument tested positively by both rules, but with a different action.*

We refer the interested reader to [1, Section 4.1] for an in-depth discussion of the constraints for smooth and distinctive operators.

Remark 2. The ternary operator f from Example 1 and the parallel composition operator from Example 2 are smooth but not distinctive. On the other hand, the classic communication merge operator [6,8], given by the rules $\dfrac{x \xrightarrow{a} x' \quad y \xrightarrow{b} y'}{x \mid y \xrightarrow{c} x' \,||\, y'}$ $(\gamma(a,b) = c)$, is smooth and distinctive. Moreover, assuming that γ is commutative, any TSS whose signature Σ includes $||$ and $\mid$ with the previously given rules is in the comm-GSOS format with respect to any set of partitions $\prod^{\Sigma}$ such that $\prod_{\mid} = \prod_{||} = \{\{1,2\}\}$.

Definition 10 (Discarding and Good Operators). *A smooth GSOS rule of the form given in Definition 9 is* discarding *if none of the variables x_i with $i \in J$ and $n_i > 0$ occurs in $C[\overrightarrow{x}, \overrightarrow{y}]$. A smooth operator is* discarding *if so are all the rules for it. A smooth operator is* good *[11] if it is both distinctive and discarding.*

In the remainder of this subsection, we assume that the GSOS system $\mathcal{T}$ has signature Σ and is in the comm-GSOS format with respect to a set of partitions $\prod^{\Sigma}$. Let $f \in \Sigma$ be a good operator that is not in the signature for BCCSP, and let n be its arity. Our goal is to generate an axiom system that can be used to turn any term of the form $f(t_1, \ldots, t_n)$, where the t_i's are in head normal form, into a head normal form. In the generation of the axiom system, we will exploit the commutativity information that is provided by the partition $\prod_f$ and therefore we assume that $n \geq 2$. (If f is either a constant or a unary operator, then it will be axiomatized exactly as in [1], since commutativity information is immaterial.) Let $I_f \subseteq [n]$ be the set of arguments that are tested positively by f and let J_f be the complement of I_f. Assume that $\prod_f = \{K_1, \ldots, K_\ell\}$. Since $\mathcal{T}$ is in the comm-GSOS format with respect to $\prod^{\Sigma}$, and f is smooth and distinctive, it is not hard to see that $K_h \subseteq I_f$ or $K_h \subseteq J_f$, for each $h \in [\ell]$. Indeed exactly one of the above inclusions holds. Let $\prod_f^+ = \{K \mid K \in \prod_f \text{ and } K \subseteq I_f\}$ and $\prod_f^- = \{K \mid K \in \prod_f \text{ and } K \subseteq J_f\}$. We use K_f^+ (respectively, K_f^-) to denote a subset of I_f (respectively, J_f) that results by choosing exactly one representative element for each $K \in \prod_f^+$ (respectively, $K \in \prod_f^-$).

Example 3. Consider the communication merge operator $|$ from Remark 2. We already remarked that, when the communication function γ is commutative, the rules for $|$ are in the comm-GSOS format with respect to any set of partitions $\prod^{\Sigma}$ such that $\prod_{|} = \prod_{||} = \{\{1,2\}\}$. For the operator $|$, we may take $K^{+}_{|} = \{1\}$. Since the rules for $|$ have no negative premises, $K^{-}_{|}$ is empty.

Definition 11. *Let f be a good n-ary operator, and let K^{+}_{f} and K^{-}_{f} be defined as above. We associate with f the finite axiom system E_f consisting of the following equations.*

1. Distributivity laws*: For each $i \in K^{+}_{f}$, we have the equation:*

$$f(x_1, \ldots, x_i + x'_i, \ldots, x_n) = f(x_1, \ldots, x_i, \ldots, x_n) + f(x_1, \ldots, x'_i, \ldots, x_n).$$

2. Peeling laws*: For each rule for f of the form given in Definition 9, each $k \in K^{-}_{f}$ with $n_k > 0$ and each $a \notin \{b_{kj} \mid 1 \leq j \leq n_k\}$, we have the equation:*

$$f(P_1, \ldots, P_n) = f(Q_1, \ldots, Q_n),$$

where $P_i \equiv \begin{cases} a_i.y_i & i \in I \\ a.x'_k + x''_k & i = k \\ x_i & \text{otherwise} \end{cases}$ *and* $Q_i \equiv \begin{cases} a_i.y_i & i \in I \\ x''_k & i = k \\ x_i & \text{otherwise.} \end{cases}$

3. Action laws*: For each rule for f of the form given in Definition 9, we have the equation:* $f(P_1, \ldots, P_n) = c.C[\overrightarrow{P}, \overrightarrow{y}]$, *where* $P_i \equiv \begin{cases} a_i.y_i & i \in I \\ \mathbf{0} & i \in J \text{ and } n_i > 0 \\ x_i & \text{otherwise.} \end{cases}$

4. Inaction laws*: For each $i \in K^{+}_{f}$, we have the equation*

$$f(x_1, \ldots, x_{i-1}, \mathbf{0}, x_{i+1}, \ldots, x_n) = \mathbf{0}.$$

Suppose that, for each $i \in [n]$, term P_i is of the form $a.z_i$ when $i \in I_f$, and of the form $a.z_i + z'_i$ or z_i when $i \in J_f$. Suppose further that, for each rule for f of the form given in Definition 9, there exists some $i \in [n]$ such that one of the following holds:
   - *$i \in I_f$ and ($P_i \equiv a.z_i$, for some $a \neq a_i$),*
   - *$i \in J_f$ and ($P_i \equiv b_{ij}.z_i + z'_i$, for some $1 \leq j \leq n_i$).*

   Then we have the equation $f(P_1, \ldots, P_n) = \mathbf{0}$.

5. Commutativity laws*: For each equivalence class $K \in \prod_f$ and each two $i, j \in K$ such that $i < j$, we have the equation:*

$$f(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n) = f(x_1, \ldots, x_j, \ldots, x_i, \ldots, x_n).$$

Theorem 2. *Consider a TSS $\mathcal{T}$ in GSOS format. Let Σ_g be a collection of good operators of $\mathcal{T}$. Let E_{Σ_g} be the finite axiom system that consists of the axioms in E_{BCCSP} and the axioms in E_f, for each $f \in \Sigma_g$. Then, for any GSOS system $\mathcal{T}'$ such that $\mathcal{T} \sqsubseteq \mathcal{T}'$, the axiom system E_{Σ_g} is sound and is ground complete for terms built using the operations in the signature Σ_g and those in the signature of BCCSP.*

Example 4. For the communication merge operator $|$, taking $K_{|}^{+} = \{1\}$ as in Example 3, Definition 11 yields the following axiom system $E_{|}$:

$$\begin{array}{ll}
\text{distributivity:} & (x+y) \mid z = (x \mid z) + (y \mid z), \\
\text{action:} & a.x \mid b.y = c.(x \,||\, y) \quad \text{if } \gamma(a,b) = c, \\
\text{inaction:} & \mathbf{0} \mid y = \mathbf{0}, \\
\text{inaction:} & a.x \mid b.y = \mathbf{0} \quad \text{if } \gamma(a,b) \text{ is undefined}, \\
\text{commutativity:} & x \mid y = y \mid x.
\end{array}$$

These are exactly the equations describing the interplay between the operator $|$ and the BCCSP operators given in Table 7.1 on page 204 of [6].

4.2 Turning Bad into Good

In order to handle arbitrary GSOS operators, one needs two additional procedures: one for transforming non-smooth operators into smooth and discarding (but not necessarily distinctive) operators, and one for expressing smooth, discarding and non-distinctive operators in terms of good operators. We adopt the same approach for the first procedure as the one presented in Lemma 4.13 in [1]. On the other hand, for the second procedure, we improve on the algorithm derived from Lemma 4.10 in [1].

The step from smooth, discarding and non-distinctive operators to good ones involves the synthesis of several new operators. We now show how to improve this transformation, as presented in the aforementioned reference, by reducing the number of the generated auxiliary operators, making use of the ideas underlying the generalized commutativity format presented in Section 3.

Making Smooth and Discarding Operators Distinctive. Consider a TSS $\mathcal{T}$ with signature Σ in the comm-GSOS format with respect to a set of partitions $\prod^{\Sigma}$. Let $f \in \Sigma$ be a smooth and discarding, but not distinctive operator, and let n be its arity. We will now show how to express f in terms of good operators. We start with partitioning the set of f-defining rules into sets $R_1, \ldots, R_m$, $m > 1$, such that f is distinctive when its rules are restricted to those in R_i for each $i \in [m]$. Note that all the rules in each R_i test the same arguments positively. If $\prod_f$ is the discrete partition over $[n]$ then one proceeds by axiomatizing f as in the version of the original algorithm based on the so-called peeling laws presented in [1]. Indeed, in that case, f has no pair of commutative arguments. Suppose therefore that $\prod_f$ is not the discrete partition, and take some $K \in \prod_f$ of maximum cardinality. (Any non-singleton K would do in what follows. However, picking a set K of maximum cardinality will reduce the number of auxiliary operators that is generated by the procedure outlined below.) Our aim now is to define when two sets of rules for f are 'essentially the same up to the commutative arguments in K' and to use this information in order to synthesize enough good operators for expressing f up to bisimilarity.

Definition 12.

– *Let d and d' be two f-defining and l-emitting rules. We say that d' is a* commutative mirror *of d with respect to K and $\prod^{\Sigma}$ if the constraints in Definition 8 are met for*

some $j, k \in K$ with $j < k$. We use $\overset{K}{\smile}$ to denote the reflexive and transitive closure of the relation 'is a commutative mirror with respect to K'.

– *Let R and R' be two sets of f-defining rules. We write $R \overset{K}{\smile} R'$ if, and only if, (1) for each $d \in R$ there is some $d' \in R'$ such that $d \overset{K}{\smile} d'$, and (2) for each $d' \in R'$ there is some $d \in R$ such that $d \overset{K}{\smile} d'$.*

Example 5. Consider the ternary operator f defined by the rules on page 39. That operator is smooth and discarding, but not distinctive. Collecting all the rules that test the same arguments positively in the same set, we obtain the following four sets of rules:

– R_1 contains all the rules of the form $\dfrac{x \xrightarrow{a} x'}{f(x,y,z) \xrightarrow{a} f(x',y,z)}$ $(a \in L)$.

– R_2 contains all the rules of the form $\dfrac{y \xrightarrow{a} y'}{f(x,y,z) \xrightarrow{a} f(x,y',z)}$ $(a \in L)$.

– R_3 contains all the rules of the form $\dfrac{x \xrightarrow{a} x',\ z \xrightarrow{a} z'}{f(x,y,z) \xrightarrow{a} f(x',y,z')}$ $(a \in L)$.

– R_4 contains all the rules of the form $\dfrac{y \xrightarrow{a} y',\ z \xrightarrow{a} z'}{f(x,y,z) \xrightarrow{a} f(x,y',z')}$ $(a \in L)$.

We have already seen in Example 1 that any GSOS system including the operator f is in the comm-GSOS format with respect to any set of partitions $\prod^{\Sigma}$ such that $\prod_f = \{\{1,2\},\{3\}\}$. Take $K = \{1,2\}$. It is not hard to see that $R_1 \overset{K}{\smile} R_2$ and $R_3 \overset{K}{\smile} R_4$ hold. Indeed, as we observed in Example 1, each a-emitting rule in R_1 (respectively, R_3) is the commutative mirror of the a-emitting rule in R_2 (respectively, R_4) with respect to K, and vice versa.

Lemma 1. *$\overset{K}{\smile}$ is an equivalence relation over f-defining rules and over sets of f-defining rules.*

Recall that $\{R_1, \ldots, R_m\}$, $m > 1$, is a partition of the set of f-defining rules such that f is distinctive when its rules are restricted to those in R_i for each $i \in [m]$. Consider $\{R_1, \ldots, R_m\}/\overset{K}{\smile}$, the quotient of the set $\{R_1, \ldots, R_m\}$ with respect to the equivalence relation $\overset{K}{\smile}$. Let $\rho_1, \ldots, \rho_\ell$ be representatives of its equivalence classes. For example, in the case of the operator considered in Example 5 above, one could pick R_1 and R_4, say, as representatives of the two equivalence classes with respect to $\overset{\{1,2\}}{\smile}$. We proceed by adding to the signature Σ fresh n-ary operator symbols $f_1, \ldots, f_\ell$. The rules for the operator f_i are obtained by simply turning those in ρ_i into f_i-defining ones. Let $\mathcal{T}'$ be the resulting disjoint extension of $\mathcal{T}$. Following [1], we now need to generate an axiom that expresses f in terms of $f_1, \ldots, f_\ell$.

Definition 13. *Let $n > 0$ and let $K \subseteq [n]$. A bijection $\pi : [n] \to [n]$ is a* K-permutation *if it is the identity function over $[n] \setminus K$.*

The equation relating f to the f_i's, $i \in [\ell]$, can now be stated as follows:

$$f(x_1, \ldots, x_n) = \sum_{i=1}^{\ell} \sum \{f_i(x_{\pi(1)}, \ldots, x_{\pi(n)}) \mid \pi \text{ is a } K\text{-permutation}\}. \qquad (1)$$

For our running example, namely the ternary operator f defined by the rules on page 39 and considered in Examples 1 and 5, with the choice of representatives mentioned above, there are two auxiliary operators f_1 and f_2 with rules $\dfrac{x \xrightarrow{a} x'}{f_1(x,y,z) \xrightarrow{a} f(x',y,z)}$ $\dfrac{y \xrightarrow{a} y',\ z \xrightarrow{a} z'}{f_2(x,y,z) \xrightarrow{a} f(x,y',z')}$, where a ranges over L. Apart from the identity function over $[3]$, the only $\{1,2\}$-permutation is the one that swaps 1 and 2. Therefore, instantiating equation (1), we obtain that

$$f(x_1,x_2,x_3) = f_1(x_1,x_2,x_3) + f_1(x_2,x_1,x_3) + f_2(x_1,x_2,x_3) + f_2(x_2,x_1,x_3).$$

Using Definition 6, the family of partitions $\prod^{\Sigma}$ can be extended to any signature that includes the signature $\Sigma \cup \{f_i \mid i \in [\ell]\}$. Note that any disjoint extension of $\mathcal{T}'$ is in the comm-GSOS format with respect to this extension of $\prod^{\Sigma}$.

Proposition 2. *Equation (1) is sound in any disjoint extension of $\mathcal{T}'$.*

Equation (1) can be simplified in case any of the auxiliary operators $f_1, \ldots, f_\ell$ is commutative in the set of arguments K. Indeed, let $N \subseteq [\ell]$, and assume that $\mathcal{T}'$ is in the comm-GSOS format with respect to the family of partitions that associates with each operator g the partition $\prod_g^{\Sigma}$ when $g \in \Sigma$, the partition $\{K\} \cup 1_{[n]\setminus K}$ when $g \in \{f_i \mid i \in N\}$, and the partition $1_{[n]}$ otherwise. Then we have the following result.

Proposition 3. *The following equation is sound in any disjoint extension of $\mathcal{T}'$.*

$$f(x_1,\ldots,x_n) = \sum_{i\in[\ell]\setminus N} \sum \{f_i(x_{\pi(1)},\ldots,x_{\pi(n)}) \mid \pi \textit{ is a } K\textit{-permutation}\} + \sum_{i\in N} f_i(x_1,\ldots,x_n) \tag{2}$$

In the following section, we will see that the above simplification leads to an axiomatization of the classic parallel composition operator that is equal to an existing handcrafted one. Of course, if either N or $[\ell] \setminus N$ are empty, the corresponding $\mathbf{0}$ summand can be omitted in equation (2).

Turning Non-Smooth Operators into Smooth Ones. The methods we have presented so far yield an algorithm that, given a TSS $\mathcal{T}$ with signature Σ in the comm-GSOS format with respect to a set of partitions $\prod^{\Sigma}$, can be used to generate a disjoint extension $\mathcal{T}'$ of $\mathcal{T}$ over some signature Σ' that includes Σ and a finite axiom system E such that E is sound modulo bisimilarity over any disjoint extension of $\mathcal{T}'$ and is head normalizing for all closed Σ'-terms. Ground-completeness of E with respect to bisimilarity over $\mathcal{T}'$ (and therefore over $\mathcal{T}$) follows using standard reasoning, by possibly using the well-known Approximation Induction Principle [8] if $\mathcal{T}'$ is not semantically well founded. See [1] for details.

The algorithm has the following steps:

1. Start with the axiom system E_{BCCSP} and consider next the operators that are not in the signature for BCCSP.
2. For each non-smooth operator $f \in \Sigma$, generate a fresh smooth and discarding operator f', and add to the axiom system the equation expressing f in terms of f' as in Lemma 4.13 in [1].
3. For each smooth and discarding, but not distinctive, operator f in the resulting signature, generate a family of fresh good operators $f_1, \ldots f_\ell$, as indicated in this section, and add to the axiom system the instance of equation (1) or of equation (2), as appropriate, expressing f in terms of $f_1, \ldots f_\ell$.
4. For each good operator in the resulting signature, add to the axiom system the equations mentioned in the statement of Theorem 2.

5 Axiomatizing Parallel Composition

Let us concretely analyze the axiomatization derived using the procedure described above for the classic parallel composition operator $||$ from Example 2. We assume henceforth that the partial synchronization function γ is commutative, so that $||$ is $\{\{1,2\}\}$-commutative. As we observed in Remark 2, the parallel composition operator is smooth but not distinctive. When we partition the set of rules for $||$ into subsets of rules that test the same arguments positively, we obtain three sets R_1, R_2 and R_3, where each R_i consists of all the instances of rule (p_i) from Example 2. It is easy to see that $R_1 \overset{\{1,2\}}{\smile} R_2$. Therefore, following the procedure described in Section 4.2, we can generate two auxiliary binary operators, which are the classic left merge and communication operators, denoted by $\between$ and $|$, respectively. The rules for $|$ are those in Remark 2 and those for the left merge operator are $\dfrac{x \xrightarrow{a} x'}{x \between y \xrightarrow{a} x' \,||\, y}$ $(a \in L)$. Since we know that $|$ is $\{\{1,2\}\}$-commutative, the relationship between $||$ and the two auxiliary operators can be expressed using equation (2), whose relevant instance becomes

Standard	Optimized
$x \,\|\| \, y = (x \between y) + (x \between y) + (x \mid y)$	$x \,\|\| \, y = (x \between y) + (y \between x) + (x \mid y)$
$(a.x) \between y = a.(x \,\|\| \, y)$	$(a.x) \between y = a.(x \,\|\| \, y)$
$x \between (a.y) = a.(x \,\|\| \, y)$	$(a.x) \mid (b.y) = c.(x \,\|\| \, y)$ if $\gamma(a,b) = c$
$(a.x) \mid (b.y) = c.(x \,\|\| \, y)$ if $\gamma(a,b) = c$	$(x + y) \between z = (x \between z) + (y \between z)$
$(x + y) \between z = (x \between z) + (y \between z)$	$(x + y) \mid z = (x \mid z) + (y \mid z)$
$x \between (y + z) = (x \between y) + (x \between z)$	
$(x + y) \mid z = (x \mid z) + (y \mid z)$	
$x \mid (y + z) = (x \mid y) + (x \mid z)$	$\mathbf{0} \between x = \mathbf{0}$
$\mathbf{0} \between x = \mathbf{0}$ $\quad x \between \mathbf{0} = \mathbf{0}$	$\mathbf{0} \mid x = \mathbf{0}$
$\mathbf{0} \mid x = \mathbf{0}$ $\quad x \mid \mathbf{0} = \mathbf{0}$	$(a.x) \mid (b.y) = \mathbf{0}$ if $\gamma(a,b)$ is undefined
$(a.x) \mid (b.y) = \mathbf{0}$ if $\gamma(a,b)$ is undefined	$x \mid y = y \mid x$

Fig. 1. Axiomatizing $||$

$$x \,||\, y = (x \between y) + (y \between x) + (x \mid y).$$

This is exactly equation M in Table 7.1 on page 204 of [6]. The axioms for $\mid$ produced by our methods are those given in Example 4. On the other hand, the left merge operator is axiomatized as in [1] since commutativity information is immaterial for it.

In Figure 1 we compare the axiomatization for the parallel composition operator $||$ derived using the algorithm from [1] and the 'optimized axiomatization' one obtains using the algorithm mentioned above. (We omit the four equations in the axiom system E_{BCCSP} recalled in Section 4.) The axioms generated by the algorithm from [1] do resemble the original axioms of [9] to a large extent. The auxiliary operator $\between$ is called right merge in the literature.

6 Conclusions and Future Work

In this paper, we have taken a first step towards marrying two lines of development within the field of the meta-theory of SOS, viz. the study of algorithms for the automatic generation of ground-complete axiomatizations for bisimilarity from SOS specifications (see, for instance, [1,7,20]) and the development of rule formats guaranteeing the validity of algebraic laws, such as those surveyed in [5]. More specifically, we have presented a rule format for commutativity that refines the one offered in [16] in that it allows one to consider various sets of commutative arguments, and we have used the information provided by that rule format to refine the algorithm for the automatic generation of ground-complete axiomatizations for bisimilarity from [1]. The resulting procedure yields axiom systems that use fewer auxiliary operators to axiomatize commutative operators than the one from [1]. Moreover, in some important cases, the mechanically produced axiomatizations of some operators are identical to the hand-crafted ones from the literature.

The ideas we have presented in this paper have never been explored before, and they enrich the toolbox one can use when reasoning about bisimilarity by means of axiomatizations. Moreover, the combination of two closely related strands of research on the meta-theory of SOS we have begun in this paper is of theoretical interest and may lead to further improvements on algorithms for the automatic generation of axiomatic characterizations of bisimilarity. As future work, we will implement the axiomatization procedure presented in this paper in the PREG Axiomatizer tool [2]. We also intend to explore the use of other rule formats for algebraic properties in improving mechanized axiomatizations for bisimilarity. The ultimate goals of this research are to make automatically generated axiomatizations comparable to the known ones from the literature and to make the first steps towards the automatic generation of axiomatizations that are complete for open terms. The latter goal is a very ambitious one since obtaining complete axiomatizations of bisimilarity is a very hard research problem even for sufficiently rich fragments of specific process calculi; see, for instance, [3].

References

1. Aceto, L., Bloom, B., Vaandrager, F.W.: Turning SOS rules into equations. Information and Computation 111, 1–52 (1994)

2. Aceto, L., Caltais, G., Goriac, E.-I., Ingolfsdottir, A.: PREG Axiomatizer - a ground bisimilarity checker for GSOS with predicates. In: Corradini, A., Klin, B., Cîrstea, C. (eds.) CALCO 2011. LNCS, vol. 6859, pp. 378–385. Springer, Heidelberg (2011)
3. Aceto, L., Fokkink, W., Ingólfsdóttir, A., Luttik, B.: Finite equational bases in process algebra: Results and open questions. In: Middeldorp, A., van Oostrom, V., van Raamsdonk, F., de Vrijer, R. (eds.) Processes... (Klop Festschrift). LNCS, vol. 3838, pp. 338–367. Springer, Heidelberg (2005)
4. Aceto, L., Fokkink, W., Verhoef, C.: Structural operational semantics. In: Bergstra, J.A., Ponse, A., Smolka, S.A. (eds.) Handbook of Process Algebra, ch. 3, pp. 197–292. Elsevier Science, Dordrecht (2001)
5. Aceto, L., Ingolfsdottir, A., Mousavi, M., Reniers, M.A.: Algebraic properties for free! Bulletin of the European Association for Theoretical Computer Science 99, 81–104 (2009)
6. Baeten, J., Basten, T., Reniers, M.: Process Algebra: Equational Theories of Communicating Processes. Cambridge Tracts in Theoretical Computer Science, vol. 50. Cambridge University Press (2009)
7. Baeten, J.J., de Vink, E.P.: Axiomatizing GSOS with termination. Journal of Logic and Algebraic Programming 60-61, 323–351 (2004)
8. Bergstra, J.A., Klop, J.W.: Fixedpoint semantics in process algebra. Technical Report IW 206/82, Center for Mathematics, Amsterdam, The Netherlands (1982)
9. Bergstra, J.A., Klop, J.W.: Process algebra for synchronous communication. Information and Control 60(1-3), 109–137 (1984)
10. Bloom, B., Istrail, S., Meyer, A.R.: Bisimulation can't be traced. Journal of the ACM 42(1), 232–268 (1995)
11. Bosscher, D.: Term rewriting properties of SOS axiomatisations. In: Hagiya, M., Mitchell, J.C. (eds.) TACS 1994. LNCS, vol. 789, pp. 425–439. Springer, Heidelberg (1994)
12. van Glabbeek, R.J.: The linear time - branching time spectrum I. In: Bergstra, J.A., Ponse, A., Smolka, S.A. (eds.) Handbook of Process Algebra, ch. 1, pp. 3–100. Elsevier Science, Dordrecht (2001)
13. Hennessy, M., Milner, A.R.: Algebraic laws for non-determinism and concurrency. Journal of the ACM 32(1), 137–161 (1985)
14. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall (1985)
15. Milner, A.R.: Communication and Concurrency. Prentice-Hall (1989)
16. Mousavi, M., Reniers, M., Groote, J.F.: A syntactic commutativity format for SOS. Information Processing Letters 93, 217–223 (2005)
17. Mousavi, M., Reniers, M.A., Groote, J.F.: SOS formats and meta-theory: 20 years after. Theoretical Computer Science 373, 238–272 (2007)
18. Park, D.M.: Concurrency and automata on infinite sequences. In: Deussen, P. (ed.) GI-TCS 1981. LNCS, vol. 104, pp. 167–183. Springer, Heidelberg (1981)
19. Plotkin, G.D.: A structural approach to operational semantics. Journal of Logic and Algebraic Progamming 60, 17–139 (2004)
20. Ulidowski, I.: Finite axiom systems for testing preorder and De Simone process languages. Theoretical Computer Science 239(1), 97–139 (2000)

Positive Fragments of Coalgebraic Logics

Adriana Balan[1], Alexander Kurz[2], and Jiří Velebil[3,⋆]

[1] University Politehnica of Bucharest, Romania
adriana.balan@mathem.pub.ro
[2] University of Leicester, UK
ak155@mcs.le.ac.uk
[3] Faculty of Electrical Engineering,
Czech Technical University in Prague, Czech Republic
velebil@math.feld.cvut.cz

Abstract. Positive modal logic was introduced in an influential 1995 paper of Dunn as the positive fragment of standard modal logic. His completeness result consists of an axiomatization that derives all modal formulas that are valid on all Kripke frames and are built only from atomic propositions, conjunction, disjunction, box and diamond.

In this paper, we provide a coalgebraic analysis of this theorem, which not only gives a conceptual proof based on duality theory, but also generalizes Dunn's result from Kripke frames to coalgebras of weak-pullback preserving functors.

For possible application to fixed-point logics, it is note-worthy that the positive coalgebraic logic of a functor is given not by all predicate-liftings but by all monotone predicate liftings.

Keywords: coalgebraic logic, duality, positive modal logic.

1 Introduction

Consider modal logic as given by atomic propositions, Boolean operations, and a unary box, together with its usual axiomatisation stating that box preserves finite meets. In [10], Dunn answered the question of an axiomatisation of the positive fragment of this logic, where the positive fragment is given by atomic propositions, lattice operations, and unary box and diamond.

Here we seek to generalize this result from Kripke frames to coalgebras for a weak pullback preserving functor. Whereas Dunn had no need to justify that the positive fragment actually *adds* a modal operator (the diamond), the general situation requires a conceptual clarification of this step. And, as it turns out, what looks innocent enough in the familiar case is at the heart of the general construction.

In the general case, we start with a functor $T : \mathsf{Set} \to \mathsf{Set}$. From T we can obtain by duality a functor $L : \mathsf{BA} \to \mathsf{BA}$ on the category BA of Boolean algebras, so that the free L-algebras are exactly the Lindenbaum algebras of the

⋆ Supported by the grant No. P202/11/1632 of the Czech Science Foundation.

R. Heckel and S. Milius (Eds.): CALCO 2013, LNCS 8089, pp. 51–65, 2013.

modal logic. We are going to take the functor L itself as the category theoretic counterpart of the corresponding modal logic. How should we construct the positive T-logic? Dunn gives us a hint in that he notes that in the same way as standard modal logic is given by algebras over BA, positive modal logic is given by algebras over the category DL of (bounded) distributive lattices. It follows that the positive fragment of (the logic corresponding to) L should be a functor $L' : \mathsf{DL} \to \mathsf{DL}$ which, in turn, by duality, should arise from a functor $T' : \mathsf{Pos} \to \mathsf{Pos}$ on the category Pos of posets and monotone maps.

The centre-piece of our construction is now the observation that any finitary-functor $T : \mathsf{Set} \to \mathsf{Set}$ has a canonical extension to a functor $T' : \mathsf{Pos} \to \mathsf{Pos}$. Theorem 4.10 then shows that this construction $T \mapsto T' \mapsto L'$ indeed gives the positive fragment of L and so generalizes Dunn's theorem.

An important observation about the positive fragment is the following: given any Boolean formula, we can rewrite it as a positive formula with negation only appearing on atomic propositions. In other words, the translation β from positive logic to Boolean logic given by

$$\beta(\Diamond\phi) = \neg\Box\neg\beta(\phi) \tag{1}$$

$$\beta(\Box\phi) = \Box\beta(\phi) \tag{2}$$

induces a bijection (on equivalence classes of formulas taken up to logical equivalence). More algebraically, we can formulate this as follows.

Given a Boolean algebra $B \in \mathsf{BA}$, let LB be the free Boolean algebra generated by $\{\Box b \mid b \in B\}$ modulo the axioms of modal logic. Given a distributive lattice A, let $L'A$ be the free distributive lattice generated by $\{\Box a : a \in A\} \cup \{\Diamond a \mid a \in A\}$ modulo the axioms of positive modal logic. Further, let us denote by $W : \mathsf{BA} \to \mathsf{DL}$ the forgetful functor. Then the above observation that every modal formula can be written, up to logical equivalence, as a positive modal formula with negations pushed to atoms, can be condensed into the statement that the (natural) distributive lattice homomorphism

$$\beta_B : L'WB \to WLB \tag{3}$$

induced by (1), (2) is an isomorphism.

Our main results are the following. If T' is an extension of T and L, L' are the induced logics, then $\beta : L'W \to WL$ exists. If, moreover, T' is the induced extension (posetification) of T and T preserves weak pullbacks, then β is an isomorphism. Furthermore, in the same way as the induced logic L can be seen as the logic of all predicate liftings of T, the induced logic L' is the logic of all *monotone* predicate liftings of T. These results depend crucially on the fact that the posetification T' of T is defined as a completion with respect to Pos-enriched colimits. On the algebraic side the move to Pos-enriched colimits guarantees that the modal operators are monotone. Accordingly, and recalling [19, Theorem 4.7] stating that a functor $L' : \mathsf{DL} \to \mathsf{DL}$ preserves ordinary sifted colimits if and only if it has a presentation by operations and equations, we show here that $L' : \mathsf{DL} \to \mathsf{DL}$ preserves *enriched* sifted colimits if and only if it has a presentation by *monotone* operations and equations. To see the relevance of a presentation

result specific to monotone operations, observe that in the example of positive modal logic it is indeed the case that both $\Box$ and $\Diamond$ are monotone.

2 On Coalgebras and Coalgebraic Logic

I. Coalgebras. A Kripke model (W, R, v) with $R \subseteq W \times W$ and $v : W \to 2^{\mathsf{AtProp}}$ can also be described as a coalgebra $W \to \mathcal{P}W \times 2^{\mathsf{AtProp}}$, where $\mathcal{P}W$ stands for the powerset of W. This point of view suggests to generalize modal logic from Kripke frames to coalgebras

$$\xi : X \to TX$$

where T may now be any functor $T : \mathsf{Set} \to \mathsf{Set}$. We get back Kripke models by putting $TX = \mathcal{P}X \times 2^{\mathsf{AtProp}}$. We also get the so-called bounded morphisms or p-morphisms as coalgebras morphisms, that is, as maps $f : X \to X'$ such that $Tf \circ \xi = \xi' \circ f$.

II. Coalgebras and Algebras. More generally, for any category $\mathcal{C}$ and functor $T : \mathcal{C} \to \mathcal{C}$, we have the category $\mathsf{Coalg}(T)$ of T-coalgebras with objects and morphisms as above. Dually, $\mathsf{Alg}(T)$ is the category where the objects $TX \xrightarrow{\alpha} X$ are arrows in $\mathcal{C}$ and where the morphisms $f : (X, \alpha) \to (X', \alpha')$ are arrows $f : X \to X'$ in $\mathcal{C}$ such that $f \circ \alpha = \alpha' \circ Tf$. It is worth noting that T-coalgebras over $\mathcal{C}$ are dual to T^{op}-algebras over $\mathcal{C}^{op}$.

III. Duality of Boolean Algebras and Sets. The abstract duality between algebras and coalgebras becomes particularly interesting if we put it on top of a concrete duality, such as the dual adjunction between the category Set of sets and functions and the category BA of Boolean algebras. We denote by $P : \mathsf{Set}^{op} \to \mathsf{BA}$ the functor taking powersets and by $S : \mathsf{BA} \to \mathsf{Set}^{op}$ the functor taking ultrafilters. Alternatively, we can describe these functors by $PX = \mathsf{Set}(X, 2)$ and $SA = \mathsf{BA}(A, \mathbb{2})$, which also determines their action on arrows (here $\mathbb{2}$ denotes the two-element Boolean algebra). P and S are adjoint, satisfying $\mathsf{Set}(X, SA) \cong \mathsf{BA}(A, PX)$. Restricting P and S to finite Boolean algebras/sets, this adjunction becomes a dual equivalence.

IV. Boolean Logics for Coalgebras, Syntax. What now are logics for coalgebras? We follow a well-established methodology in modal logic ([6]) and study modal logics via the associated category of modal algebras. More formally, given a modal logic $\mathcal{L}$ extending Boolean propositional logic and with associated category $\mathcal{A}$ of modal algebras, we describe $\mathcal{L}$ by a functor

$$L : \mathsf{BA} \to \mathsf{BA}$$

so that the category $\mathsf{Alg}(L)$ of algebras for the functor L coincides with $\mathcal{A}$. In particular, the Lindenbaum algebra of $\mathcal{L}$ will be the initial L-algebra.

Example 2.1. Let $T = \mathcal{P}$ be the powerset functor and $L : \mathsf{BA} \to \mathsf{BA}$ be the functor mapping an algebra A to the algebra LA generated by $\Box a$, $a \in A$, and quotiented by the relation stipulating that $\Box$ preserves finite meets, that is,

$$\Box\top = \top \qquad \Box(a \wedge b) = \Box a \wedge \Box b \tag{4}$$

$\mathsf{Alg}(L)$ is the category of modal algebras (Boolean algebras with operators), a result which appears to be explicitly stated first in [1].

V. Boolean Logics for Coalgebras, Semantics. The semantics of such a logic is described by a natural transformation

$$\delta : LP \to PT^{op}$$

Intuitively, each modal operator in LPX is assigned its meaning as a subset of TX. More formally, δ allows us to lift $P : \mathsf{Set}^{op} \to \mathsf{BA}$ to a functor $P^\sharp : \mathsf{Coalg}(T) \to \mathsf{Alg}(L)$, and if we take a formula ϕ to be an element of the initial L-algebra (the Lindenbaum algebra of the logic), then the semantics of ϕ as a subset of a coalgebra (X, ξ) is given by the unique arrow from that initial algebra to $P^\sharp(X, \xi)$.

Example 2.2. We define the semantics $\delta_X : LPX \to P\mathcal{P}^{op}X$ by, for $a \in PX$,

$$\Box a \mapsto \{b \in \mathcal{P}X \mid b \subseteq a\}. \tag{5}$$

It is an old result in domain theory that δ_X is an isomorphism for finite X ([1]). This implies completeness of the axioms (4) with respect to Kripke semantics.

VI. Functors having Presentations by Operations and Equations. One might ask when a functor $L : \mathsf{BA} \to \mathsf{BA}$ can legitimately be considered to give rise to a modal logic. For us, in this paper, a minimal requirement on L is that $\mathsf{Alg}(L)$ is a variety in the sense of universal algebra, that is, that $\mathsf{Alg}(L)$ can be described by operations and equations, the operations then corresponding to modal operators and the equations to axioms. This happens if L is determined by its action on finitely generated free algebras (see [19]). These functors are also characterized as functors having presentations by operations and equations, or as functors preserving sifted colimits. Most succinctly, they are precisely those functors that arise as left Kan-extensions along the inclusion functor of the full subcategory of BA consisting of free algebras on finitely many generators.

VII. The (finitary, Boolean) Coalgebraic Logic of a Set*-Functor.* The general considerations laid out above suggest to define the finitary (Boolean) coalgebraic logic associated to a given functor $T : \mathsf{Set} \to \mathsf{Set}$ as

$$\mathbf{L}Fn = PT^{op}SFn \tag{6}$$

where Fn denotes the free Boolean algebra over n generators, for n ranging over natural numbers. The semantics $\boldsymbol{\delta}$ is given by observing that natural transformations $\boldsymbol{\delta} : \mathbf{L}P \to PT$ are in bijection with natural transformations

$$\hat{\boldsymbol{\delta}} : \mathbf{L} \to PT^{op}S \tag{7}$$

so that we can define $\hat{\boldsymbol{\delta}}$ to be the identity on finitely generated free algebras.

More explicitly, $\mathbf{L}A$ can be represented as the free BA over $\{\sigma(a_1, \dots a_n) \mid \sigma \in PT^{op}SFn, a_i \in A, n < \omega\}$ modulo appropriate axioms, with $\boldsymbol{\delta}_X : \mathbf{L}PX \to PT^{op}X$ given by $\boldsymbol{\delta}\sigma(a_1, \dots a_n) = PT^{op}(\hat{a})(\sigma)$ where $\hat{a} : X \to SFn$ is the adjoint transpose of $(a_1, \dots a_n) : n \to UPX$, with the forgetful functor $U : \mathsf{BA} \to \mathsf{Set}$ being right adjoint of F. [1] Of course, in concrete examples one is often able to obtain much more succinct presentations:

Proposition 2.3. *With $T = \mathcal{P}$, the functor $\mathbf{L}$ defined by* (6) *is isomorphic to the functor L of Example 2.1.*

VIII. Positive Coalgebraic Logic. It is evident that, at least for some of the developments above, not only the functor T, but also the categories Set and BA can be considered parameters. Accordingly, one expects that positive coalgebraic logic takes place over the category DL of (bounded) distributive lattices which in turn, is part of an adjunction $P' : \mathsf{Pos}^{op} \to \mathsf{DL}$, taking upsets, and $S' : \mathsf{DL} \to \mathsf{Pos}^{op}$, taking prime filters, or, equivalently, $P'X = \mathsf{Pos}(X, \mathbb{2})$ and $S'A = \mathsf{DL}(A, \mathbb{2})$ where $\mathbb{2}$ is, as before, the two-chain (possibly considered as a distributive lattice). Consequently, the 'natural semantics' of positive logics is 'ordered Kripke frames'. That is, we may define a logic for T'-coalgebras, with $T' : \mathsf{Pos} \to \mathsf{Pos}$, to be given by a natural transformation

$$\boldsymbol{\delta}' : \mathbf{L}'P' \to P'T'^{op} \tag{8}$$

where

$$\mathbf{L}'F'n = P'T'^{op}S'F'n \tag{9}$$

is a functor determined by its action on finitely generated free distributive lattices and $\boldsymbol{\delta}'$ is given by its transpose in the same way as (7).

Example 2.4. Let T' be the convex powerset functor $\mathcal{P}'$ and $L' : \mathsf{DL} \to \mathsf{DL}$ be the functor mapping a distributive lattice A to the distributive lattice $L'A$ generated by $\Box a$ and $\Diamond a$ for all $a \in A$, and quotiented by the relations stipulating that $\Box$ preserves finite meets, $\Diamond$ preserves finite joins, and

$$\Box a \wedge \Diamond b \leq \Diamond(a \wedge b) \qquad \Box(a \vee b) \leq \Diamond a \vee \Box b \tag{10}$$

The natural transformation $\delta'_X : L'P'X \to P'\mathcal{P}'^{op}X$ is defined by, for $a \in P'X$,

$$\Diamond a \mapsto \{b \in \mathcal{P}X \mid b \cap a \neq \emptyset\}, \tag{11}$$

the clause for $\Box a$ being the same as in (5).

[1] Since elements in $PTSFn$ are in one-to-one correspondence with natural transformations $\mathsf{Set}(-, 2^n) \to \mathsf{Set}(T-, 2)$, also known as predicate liftings [25], we see that the logic L coincides with the logic of all predicate liftings of [27], with the difference that L also incorporates axioms. The axioms are important to us as otherwise the natural transformation β mentioned in the introduction might not exist.

Remark 2.5. $\mathsf{Alg}(L')$ is the category of positive modal algebras of Dunn [10] and we will show that it is isomorphic to $\mathsf{Alg}(\mathbf{L}')$ in Corollary 3.6. Again we have that for finite X, δ'_X is an isomorphism, a representation first stated in [12,13], the connection with modal logic being given by [30,26,1] and investigated from a coalgebraic point of view in [24].

3 On Pos and Pos-Enriched Categories

I. The Category Pos *of Posets and Monotone Maps.* Pos is complete and cocomplete (even locally finitely presentable [3]), limits being computed as in Set, while for colimits one has to quotient the corresponding colimits obtained in the category of preordered sets and monotone maps (however, directed colimits are computed as in Set, see [3]). Pos is also cartesian closed, with the internal hom $[X, Y]$ being the poset of monotone maps from X to Y, ordered pointwise.

This paper will consider categories *enriched* in Pos because this automatically takes care of the algebraic operations being monotone. Therefore when we say category, functor, natural transformation in what follows, we always mean the enriched concept. Thus a category has ordered homsets and functors are locally monotone, that is, they preserve the order on the homsets.

When we want to deal with non-enriched concepts, we always call them *ordinary*. Thus, for example, the category Pos has its underlying ordinary category Pos_o. Everything below with the subscript o is the underlying ordinary thing of the Pos-enriched thing.

In particular, we consider Set as discretely enriched over Pos. Then $D : \mathsf{Set} \to \mathsf{Pos}$, the discrete functor, is trivially Pos-enriched. There are two more Pos-categories appearing in this paper, namely BA and DL. The first one is considered discretely enriched, while in DL the enrichment is a consequence of the natural order induced by operations.

II. Sifted Weights and Sifted (co)Limits. The theory of (locally monotone) Pos-functors and their logics of monotone modal operators naturally leads to the world of ordered varieties. Since the details are only needed for the proofs (which had to be omitted for reasons of space) we note here only that our arguments are based on [21,8,22,17].

In the non-enriched setting, a functor on a variety preserves ordinary sifted colimits iff it preserves filtered colimits and reflexive coequalizers. In the Pos-enriched setting, a functor on an ordered variety preserves (enriched) sifted colimits iff it preserves filtered colimits and reflexive coinserters. We recall that the coinserter ([16]) of a parallel pair of arrows $X \overset{f}{\underset{g}{\rightrightarrows}} Y$ in a Pos-category consists of an object $\mathsf{coins}(f, g)$ and of an arrow $\pi : Y \to \mathsf{coins}(f, g)$ with $\pi f \leq \pi g$, with the following universal property: for any $q : Y \to Z$ with $qf \leq qg$, there is a unique $h : \mathsf{coins}(f, g) \to Z$ with $h\pi = q$. Moreover, this assignment is monotone, in the sense that given $q, q' : Y \to Z$ with $q \leq q'$, $qf \leq qg$ and $q'f \leq q'g$, the

corresponding unique arrows $h, h' : \mathsf{coins}(f, g) \to Z$ satisfy $h \leq h'$. The coinserter is called reflexive if f and g have a common right inverse. By switching the arrows, one obtain the dual notion of a (coreflexive) inserter.

III. Functors Preserving Sifted Colimits and their Equational Presentation. Denote by Set_f the category of finite sets and maps and by ι the composite $\mathsf{Set}_f \hookrightarrow \mathsf{Set} \xrightarrow{D} \mathsf{Pos}$. Then Pos is the free cocompletion of Set_f under (enriched) sifted colimits [21].

A functor $\mathcal{T} : \mathsf{Pos} \to \mathsf{Pos}$ is called *strongly finitary* if one of the equivalent conditions below holds: (i) $\mathcal{T}$ is isomorphic to the left Kan extension along ι of its restriction, that is $\mathcal{T} \cong \mathsf{Lan}_\iota(\mathcal{T}\iota)$; (ii) $\mathcal{T}$ preserves sifted colimits.

Recall that there are monadic (enriched) adjunctions $F \dashv U : \mathsf{BA} \to \mathsf{Set}$, $F' \dashv U' : \mathsf{DL} \to \mathsf{Pos}$, where U and U' are the corresponding forgetful functors. We denote by $\mathbf{J} : \mathsf{BA}_{ff} \to \mathsf{BA}$ and $\mathbf{J}' : \mathsf{DL}_{ff} \to \mathsf{DL}$ the inclusion functors of the full subcategories spanned by the algebras which are free on finite (discrete po)sets.

Lemma 3.1. $\mathbf{J}$ *and* $\mathbf{J}'$ *exhibit* BA*, respectively* DL*, as the free cocompletions under sifted colimits of* BA_{ff} *and* DL_{ff}*. In particular, these functors are dense.*

Corollary 3.2. *A functor* $L : \mathsf{BA} \to \mathsf{BA}$ *has the form* $\mathrm{Lan}_{\mathbf{J}}(L\mathbf{J})$ *iff it preserves (ordinary) sifted colimits. A functor* $L' : \mathsf{DL} \to \mathsf{DL}$ *has the form* $\mathrm{Lan}_{\mathbf{J}'}(L'\mathbf{J}')$ *iff it preserves sifted colimits.*

Theorem 3.3. *Suppose* $L : \mathsf{BA} \to \mathsf{BA}$ *and* $L' : \mathsf{DL} \to \mathsf{DL}$ *preserve sifted colimits. Then they both have an equational presentation.*

Remark 3.4. The (proof of the) above theorem actually shows that every functor $L' : \mathsf{DL}_{ff} \to \mathsf{DL}$ (i.e., every L' preserving sifted colimits) has a presentation in the form of a coequalizer

$$\widehat{H_\Gamma} \rightrightarrows \widehat{H_\Sigma} \longrightarrow L'$$

for some strongly finitary signatures Γ and Σ, i.e. some locally monotone functors $\Gamma, \Sigma : |\mathsf{Set}_f| \to \mathsf{Pos}$, where $|\mathsf{Set}_f|$ is the skeleton of the category of finite sets. Here, $\widehat{H_\Sigma}$ is defined as follows: given $\Sigma : |\mathsf{Set}_f| \to \mathsf{Pos}$, $H_\Sigma : \mathsf{Set}_f \to \mathsf{Pos}$ is the *polynomial strongly finitary functor*

$$H_\Sigma n = \coprod_{k \in |\mathsf{Set}_f|} \mathsf{Set}_f(k, n) \bullet \Sigma k$$

and it extends to a strongly finitary $H_\Sigma : \mathsf{Pos} \to \mathsf{Pos}$ by sifted colimits. In the above formula, $\coprod$ and $\bullet$ refers to the coproduct, respectively copower in the category Pos. The resulting $\widehat{H_\Sigma} : \mathsf{DL}_{ff} \to \mathsf{DL}$ is thus given, at a free distributive lattice with finite discrete set of generators, by

$$\widehat{H_\Sigma}(F'Dn) = F'H_\Sigma U'(F'Dn)$$

(see Remark 3.16 of [28]) and, again, it is extended to an endofunctor on DL by means of sifted colimits.

We say that a functor $\mathsf{DL} \to \mathsf{DL}$ has a presentation by monotone operations and equations if it has a presentation by operations and equations in the sense of [7], such that, moreover, all operations are monotone. We then obtain the following enriched version of [19, Theorem 4.7], characterizing *enriched* sifted colimits preserving functors in terms of presentations with *monotone* operations.

Corollary 3.5. *A functor $L' : \mathsf{DL} \to \mathsf{DL}$ has a presentation by monotone operations and equations if and only if L' is the* Pos*-enriched left Kan extension of its restriction to finitely generated free distributive lattices.*

As in Proposition 2.3, we now obtain that

Corollary 3.6. *If T' is the the convex powerset functor, then the functor L' of Example 2.4 is isomorphic to the sifted colimits preserving functor $\mathbf{L}'$ whose restriction to DL_{ff} is $P'T'^{op}S'$ as in* (8)*.*

IV. The Pos*-extension of a* Set*-functor.* In order to relate Set and Pos-functors, we recall from [4] the following

Definition 3.7. *Let T be an endofunctor on* Set*. A* Pos*-endofunctor T' is said to be a* Pos-extension *of T if it is locally monotone and if the square*

$$\begin{array}{ccc} \mathsf{Pos} & \xrightarrow{T'} & \mathsf{Pos} \\ {\scriptstyle D}\uparrow & \nwarrow\alpha & \uparrow{\scriptstyle D} \\ \mathsf{Set} & \xrightarrow[T]{} & \mathsf{Set} \end{array} \tag{12}$$

commutes up to an isomorphism $\alpha : DT \to T'D$.

A Pos*-extension T' is called the* posetification *of T if the above square exhibits T' as $\mathsf{Lan}_D DT$ (in the* Pos*-enriched sense), having α as its unit.*

If T is finitary, then its posetification does exist. This can be seen by expressing $\mathsf{Lan}_D(DT)$ as a coend

$$\mathsf{Lan}_D(DT)X = \int^{S\in\mathsf{Set}} [DS, X] \bullet DTS \tag{13}$$

and taking into account that T is determined by its action on finite sets: explicitly, the coend becomes

$$\mathsf{Lan}_D(DT)X = \int^{n\in\mathsf{Set}_f} [Dn, X] \bullet DTn \tag{14}$$

which in turn is the following Pos-coequalizer

$$\coprod_{m,n<\omega} \mathsf{Set}(m,n) \times Tm \times [Dn, X] \rightrightarrows \coprod_{n<\omega} Tn \times [Dn, X] \xrightarrow{\pi} \mathsf{Lan}_D(DT)X \tag{15}$$

In formulae above, $[-,-]$ denotes the internal hom in Pos and $\times$ refers to the (cartesian) product of posets, ordered component-wise.

Posetifications of (finitary) Set-functors are immediate examples of strongly finitary Pos-functors. Briefly, one can say that a Pos-functor is a posetification if it has a presentation by monotone operations and discrete arities. In fact, we can be much more precise: a functor $T' : \mathsf{Pos} \to \mathsf{Pos}$ is the posetification of a finitary Set-functor if it is strongly finitary and preserves discrete sets.

Example 3.8. 1. Let $T = \mathrm{Id}$ on Set. Then the discrete connected components functor and the upper-sets-functor are both extensions of T, while $\mathrm{Id} : \mathsf{Pos} \to \mathsf{Pos}$ is the posetification (recall that the discrete functor D is dense, see [9]).

2. If we take $T = \mathcal{P}_f$ to be the (finite) power-set functor, then its posetification is the (finitely generated) convex power-set functor, with the Egli-Milner order ([4,28]).

3. The collection of (finitary) Kripke polynomial Set-functors is inductively defined as follows: $T ::= \mathrm{Id} \mid T_{X_0} \mid T_0 + T_1 \mid T_0 \times T_1 \mid T^A \mid \mathcal{P}_f$, where T_{X_0} denotes the constant functor to the set X_0; $T_0 + T_1$ is the coproduct functor $X \mapsto T_0X + T_1X$; $T_0 \times T_1$ the product functor; and T^A denotes the exponent functor $X \mapsto (TX)^A$, with A finite.
   We have just seen above that the posetification of the identity functor is again the identity, while for the constant functor T_{X_0} is an easy exercise to check that the posetification is again a constant functor, this time to the discrete poset DX_0; the posetification of the coproduct functor $T_0 + T_1$ maps a poset X to the coproduct (in the category of posets) $T_0'X + T_1'X$, where T_0' and T_1' denote the posetifications of T_0, respectively T_1; and similarly for the product and exponent functors.

4. The finitary distribution functor is given by $\mathcal{D}X = \{d : X \to [0,1] \mid \sum_{x \in X} d(x) = 1,\ \mathsf{supp}(d) < \infty\}$, where $\mathsf{supp}(d) = \{x \in X \mid d(x) \neq 0\}$. For function $f : X \to Y$, we have $\mathcal{D}(f)(d)(y) = \sum y = f(x)d(x)$. Recall that $\mathcal{D}$ preserves weak pulbacks ([29]), thus the posetification $\mathcal{D}'$ can be described using the relation lifting ([4]). Explicitly, for a poset $(X, \leq)$, $\mathcal{D}'(X, \leq)$ has underlying set $\mathcal{D}X$, where for $d, d' \in \mathcal{D}$, the partial order reads $d \leq d'$ iff there is some $\omega \in \mathcal{D}(X \times X)$ such that $\omega(x, y) > 0 \Rightarrow x \leq y$, $\sum_{y \in X} \omega(x, y) = d(x)$ and $\sum_{x \in X} \omega(x, y) = d'(y)$.

V. Morphisms of Logical Connections. We recall the (enriched) logical connections (dual adjunctions, see [20]) between sets and Boolean algebras, and between posets and distributive lattices. Both should be seen as Pos-enriched, where for the first logical connection the enrichment is discrete. They are related as follows:

$$\begin{array}{ccc} \mathsf{Set}^{op} & \underset{P}{\overset{S}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} & \mathsf{BA} \\ {\scriptstyle D^{op}}\downarrow & & \downarrow{\scriptstyle W} \\ \mathsf{Pos}^{op} & \underset{P'}{\overset{S'}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} & \mathsf{DL} \end{array} \tag{16}$$

In the top row of the above diagram, recall again that P is the contravariant powerset functor, while S maps a Boolean algebra to its set of ultrafilters. The bottom row has P' mapping a poset to the distributive lattice of its upper-sets, and S' associating to each distributive lattice the poset of its prime filters. About the pair of functors connecting the two logical connections: D was introduced earlier as the discrete functor, while W is the functor associating to each Boolean algebra its underlying distributive lattice.

It is easy to see that the pair (D^{op}, W) is a *morphism of adjunctions* in the sense of [23]. This means that the equalities

$$P'D^{op} = WP, \quad D^{op}S = S'W, \quad \epsilon' D^{op} = D^{op}\epsilon \tag{17}$$

hold, where ϵ and ϵ' are the counits of $S \dashv P$ and $S' \dashv P'$, respectively.

4 Positive Coalgebraic Logic

We now expand the propositional logics BA and DL by modal operators. We start with a Set-endofunctor T in the top left-hand corner of (16). We are mostly interested in the case where $T' : \mathsf{Pos} \to \mathsf{Pos}$ is the posetification of T (Definition 3.7) and $L : \mathsf{BA} \to \mathsf{BA}$ and $L' : \mathsf{DL} \to \mathsf{DL}$ are (the functors of) the associated logics as in (6) and (9), in which case we denote the logics by boldface letters $\mathbf{L}$ and $\mathbf{L}'$.

But some of the following holds under the weaker assumptions that T' is some extension of T and that L and L' are some logics for, respectively, T and T'. We therefore let T be a Set-endofunctor and T' be an extension of T to Pos as in (12). Logics for T, T' are given by functors $L : \mathsf{BA} \to \mathsf{BA}$ and $L' : \mathsf{DL} \to \mathsf{DL}$ and natural transformations

$$\delta : LP \to PT^{op} \qquad \delta' : L'P' \to P'T'^{op}.$$

Intuitively, δ and δ' assign to the syntax given by (presentations of) L and L' the corresponding semantics in subsets or upper sets. To compare L and L' we need the isomorphism $\alpha : DT \to T'D$ saying that T' extends T, and also the relation $WP = P'D$ from (17) (which formalizes the trivial observation that taking upsets of a discrete set is the same as taking all subsets). Referring back to the introduction, we now make the following

Definition 4.1. *We say that a logic (L', δ') for T' is* a positive fragment *of the logic (L, δ) for T, if there is a natural transformation $\beta : L'W \to WL$ with $W\delta \circ \beta P = P'\alpha^{op} \circ \delta' D^{op}$, or, in diagrams*

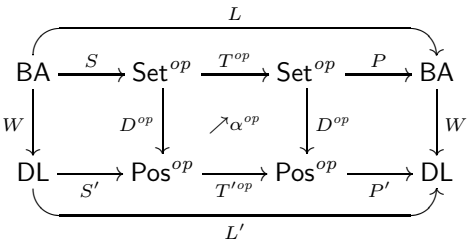
$$\begin{array}{ccccc}
\mathsf{Set}^{op} & \xrightarrow{P} & \mathsf{BA} & \xrightarrow{W} & \mathsf{DL} \\
{\scriptstyle T^{op}}\downarrow & \swarrow\delta & {\scriptstyle L}\downarrow & \swarrow\beta & \downarrow{\scriptstyle L'} \\
\mathsf{Set}^{op} & \xrightarrow[P]{} & \mathsf{BA} & \xrightarrow[W]{} & \mathsf{DL}
\end{array}
\quad = \quad
\begin{array}{ccccc}
\mathsf{Set}^{op} & \xrightarrow{D^{op}} & \mathsf{Pos}^{op} & \xrightarrow{P'} & \mathsf{DL} \\
{\scriptstyle T^{op}}\downarrow & \swarrow\alpha^{op}T'^{op} & \downarrow & \swarrow\delta' & \downarrow{\scriptstyle L'} \\
\mathsf{Set}^{op} & \xrightarrow[D^{op}]{} & \mathsf{Pos}^{op} & \xrightarrow[P']{} & \mathsf{DL}
\end{array} \tag{18}$$

We call (L', δ') the (maximal) positive fragment *of (L, δ) if β is an isomorphism.*

Recall that we defined the logics $\mathbf{L}, \mathbf{L}'$ induced by T and an extension T' as $\mathbf{L} = PTS$ and $\mathbf{L}' = P'T'^{op}S'$ on finitely generated free objects. Our desired result is to prove that a certain canonically given $\beta : \mathbf{L}'W \to W\mathbf{L}$ is an isomorphism. The difficulty, as well as the need for the proviso that T preserves weak pullbacks, stems from the fact that in DL (as opposed to BA) the class of functors determined on finitely generated free algebras is strictly smaller than the class of functors determined on finitely presentable (=finite) algebras. As stepping stones, therefore, we first investigate what happens in the cases where the functors L, L' are determined on all algebras and on finitely presentable algebras, before we turn to the situation of functors determined on strongly finitely presentable (=finitely generated free) algebras.

I. The Case of $L' = P'T'^{op}S'$ on All Algebras. We shall associate to any extension $\alpha : DT \to T'D$ the pairs (L, δ) and (L', δ') corresponding to T and T' respectively, with $L = PT^{op}S$ and $\delta = PT^{op}\epsilon : PT^{op}SP \to PT^{op}$, $L' = P'T'^{op}S'$ and δ' being defined analogously. Now the following is a consequence of (D^{op}, W) being a morphism of adjunctions (see (17)). We then immediately obtain an isomorphism β:

Proposition 4.2. *Given an extension $\alpha : DT \to T'D$, the isomorphism*



exhibits $L' = P'T'^{op}S'$ as the maximal positive fragment of $L = PT^{op}S$.

II. The Case of $\bar{L}' = P'T'^{op}S'$ on Finitely Presentable Algebras. A similar result holds if we define logics via $PT^{op}SA$ for finitely presentable A, as we are going to show now. To this end, we use the subscript $(-)_f$ to denote the restriction to finite[2] objects as e. g. when writing the dense inclusions $I : \mathsf{Set}_f \to \mathsf{Set}$, $I' : \mathsf{Pos}_f \to \mathsf{Pos}$, $J : \mathsf{BA}_f \to \mathsf{BA}$ and $J' : \mathsf{DL}_f \to \mathsf{DL}$. Note that we have the following commuting diagram

$$\begin{array}{ccc} S_f \dashv P_f & \xrightarrow{(D_f^{op}, W_f)} & S'_f \dashv P'_f \\ {\scriptstyle (I^{op}, J)}\downarrow & & \downarrow{\scriptstyle (I'^{op}, J')} \\ S \dashv P & \xrightarrow[(D^{op}, W)]{} & S' \dashv P' \end{array} \tag{19}$$

[2] As Pos is locally finitely presentable as closed category, and ordinary categories $\mathsf{Set}_o, \mathsf{DL}_o, \mathsf{BA}_o$ are also locally finitely presentable, it follows that the finitely presentable objects in all the above categories are precisely the same as in the ordinary case, i.e. the ones for which the underlying set is finite.

in the category of transformations of adjoints.

Define $(\bar{L}, \bar{\delta})$ for T as $\bar{L} = \mathsf{Lan}_J(PT^{op}SJ)$ and $\bar{\delta} = \bar{L}P \to PT^{op}$ as the adjoint transpose of $\bar{L} \to PT^{op}S$ arising from the universal property of the left Kan extension $\bar{L}$. By construction, $\bar{L}$ is finitary and is given by $PT^{op}S$ on finite(ly presentable) Boolean algebras. Similarly, obtain $(\bar{L}', \bar{\delta}')$ for T'.

Since W is left adjoint,[3] $\mathrm{Lan}_J(PT^{op}SJ)$ is preserved by W. Thus, to define an (iso)morphism $\bar{\beta} : \bar{L}'W = \mathsf{Lan}_{J'}(P'T'^{op}S'J')W \to W\bar{L} = \mathsf{Lan}_J(PT^{op}SJ)W$, it suffices to take the restriction along J of the isomorphism of Proposition 4.2, namely $\bar{\beta}_f : L'J'W_f = P'T'^{op}S'J'W_f \cong WPT^{op}SJ$.

Recall the definition of $\mathbf{L}$ from (6). Since every finitely presentable non-trivial Boolean algebra is a retract of a finitely generated free algebra, we can take $\mathbf{L} = \bar{L}$ (see eg [19, Prop 3.4]. To summarize, we have

Proposition 4.3. *The isomorphism $\bar{\beta}$ exhibits $\bar{L}' = P'T'^{op}S'J'$ as the maximal positive fragment of $(\mathbf{L}, \delta)$.*

The proposition does not yet give us the desired result, as $\bar{L}'$ is not necessarily determined by its action on finitely generated free algebras and, therefore, need not give rise to a variety of modal algebras. Paragraph III. will investigate when $\bar{L}'$ does actually have this additional property.

III. The Case of $\mathbf{L}' = P'T'^{op}S'$ on Finitely Generated Free Algebras. Recall that we denoted by $\mathbf{J} : \mathsf{BA}_{ff} \to \mathsf{BA}$ and $\mathbf{J}' : \mathsf{DL}_{ff} \to \mathsf{DL}$ the inclusion functors of the full subcategories spanned by the algebras which are free on finite discrete posets.

Definition 4.4. *Let T' be a* Pos*-endofunctor. We define the logic for T' to be the pair $(\mathbf{L}', \boldsymbol{\delta}')$, where:*

- *$\mathbf{L}' : \mathsf{DL} \to \mathsf{DL}$ is a* Pos*-functor preserving sifted colimits, whose restriction to free finitely generated distributive lattices is $P'T'^{op}S'\mathbf{J}'$, that is, $\mathbf{L}' = \mathsf{Lan}_{\mathbf{J}'}(P'T'^{op}S'\mathbf{J}')$.*
- *$\boldsymbol{\delta}' : \mathbf{L}'P' \to P'T'^{op}$ is the adjoint transpose of $\mathbf{L}' \to P'T'^{op}S'$ given by the universal property of the left Kan extension $\mathbf{L}'$.*

Remark 4.5. By the above definition, $\mathbf{L}'$ preserves sifted colimits. Thus, by Corollary 3.5, $\mathbf{L}'$ has an equational presentation by monotone operations, which in turn gives rise to a positive modal logic concretely given in terms of modal operators and axioms.

Recall that $\bar{L}' = P'T'^{op}S'$ on finitely presentable (=finite) algebras and that $\mathbf{L}' = P'T'^{op}S'$ on finitely generated free algebras.

Theorem 4.6. *Let T be a* Set*-endofunctor and T' a* Pos*-extension of T which* preserves coreflexive inserters. *Then $(\bar{L}', \bar{\delta}')$ and $(\mathbf{L}', \boldsymbol{\delta}')$ coincide. In particular, it follows that $\mathbf{L}'$ is* the maximal positive fragment *of $\mathbf{L}$.*

[3] The (enriched) right adjoint of W sends a distributive lattice A to the Boolean algebra of complemented elements in A (also known as the center of A).

Remark 4.7. The isomorphism $(\bar{L}, \bar{\delta}) \cong (\mathbf{L}, \boldsymbol{\delta})$ of the corresponding Boolean logic for Set-functors was established in [19]. (Recall that $\mathbf{L}$ was introduced in (6), while $\bar{L}$ appeared in Paragraph II. above.)

Proposition 4.8. *If T' is a Pos-endofunctor (thus locally monotone) which preserves exact squares, then it preserves embeddings and coreflexive inserters.*

The reader should think of an exact square as being the Pos-enriched analogue of a weak pullback (see [11], [5] or [4] for the precise definition).

Proposition 4.9 ([4]). *Let T be any finitary Set-functor and T' its posetification. Then T preserves weak pullbacks if and only if T' preserves exact squares.*

As a consequence of all the results of this section, we obtain

Theorem 4.10. *Let $T : \mathsf{Set} \to \mathsf{Set}$ be a finitary weak-pullback preserving functor and $T' : \mathsf{Pos} \to \mathsf{Pos}$ its posetification. Let $(\mathbf{L}, \boldsymbol{\delta})$ and $(\mathbf{L}', \boldsymbol{\delta}')$ be the associated logics of T and T', that is $\mathbf{L} = \mathrm{Lan}_{\mathbf{J}}(PT^{op}S\mathbf{J})$ and $\mathbf{L}' = \mathrm{Lan}_{\mathbf{J}'}(P'T'^{op}S'\mathbf{J}')$. Then $(\mathbf{L}', \boldsymbol{\delta}')$ is the maximal positive fragment of $(\mathbf{L}, \boldsymbol{\delta})$.*

Example 4.11. For $T = \mathrm{Id}$, the corresponding finitary logics is $\mathbf{L} = \mathrm{Id}$ on BA, with trivial semantics $\boldsymbol{\delta} : LP \to PT^{op}$. It allows the extension $T' = DC$, the discrete connected components functor. Notice that T' does not preserve embeddings, neither coreflexive inserters. The corresponding logic $\mathbf{L}'$ is given by the constant functor to the distributive lattice $\mathbb{2}$. Thus $\beta : \mathbf{L}'W \to W\mathbf{L}$ fails to be an isomorphism (it is just the unique morphism from the initial object).

Our introductory example of positive modal logic is now regained as an instance of this theorem.[4] It can also easily be adapted to Kripke polynomial functors. More interesting is the case of the probability distribution functor. We know from the theorem above that it has a maximal positive fragment, but an explicit description still needs to be worked out.

5 Monotone Predicate Liftings

In this section we show that the logic of the posetification T' of T coincides with the logic of all monotone predicate liftings of T.

Recall that a predicate lifting [25,27] of arity n for T is an ordinary natural transformation $\heartsuit : \mathsf{Set}_o(-, 2^n) \to \mathsf{Set}_o(T-, 2)$,[5] or, using the ordinary adjunction $D_o \dashv V : \mathsf{Pos}_o \to \mathsf{Set}$, an ordinary natural transformation

$$\heartsuit : \mathsf{Pos}_o(D_o-, [n, \mathbb{2}]) \to \mathsf{Pos}_o(D_oT-, \mathbb{2})$$

[4] A minor issue here is that modal logic usually takes as semantics coalgebras for the (non-finitary) powerset, whereas for the posetification to exist we sofar assumed T to be finitary. There are two solutions to this. One is to note that going from T to its finitary coreflection T_ω and then to its posetification T'_ω does not change the functors $\mathbf{L}, \mathbf{L}'$ on the algebraic side. The second is to prove that the posetification exists despite the functor not being accessible.

[5] Equivalently, it can be described as an element $\heartsuit \in \mathsf{Set}(T(2^n), 2)$.

It is called *monotone* if it lifts to a natural transformation

$$\heartsuit : \mathsf{Pos}(D-, [Dn, \mathbb{2}]) \to \mathsf{Pos}(DT-, \mathbb{2})$$

By identifying a predicate lifting with an map $\heartsuit : T(2^n) \to 2$, the above says that $\heartsuit$ is monotone if for all $\overline{a_1} \leq \overline{a_2} : D_o X \to [D_o n, \mathbb{2}]$, we have that $\overline{\heartsuit \circ Ta_1} \leq \overline{\heartsuit \circ Ta_2}$, where $\overline{f} : D_o X \to Y$ denotes the adjoint transpose of $f : X \to VY$.

Consider now a Pos-functor T' (locally monotone!) and a finite poset p. By mimicking the above, we define a predicate lifting for T' of arity p as being a natural transformation[6]

$$\heartsuit : \mathsf{Pos}(-, [p, \mathbb{2}]) \to \mathsf{Pos}(T'-, \mathbb{2})$$

Proposition 5.1. *Let T be a Set-functor and $T' : \mathsf{Pos} \to \mathsf{Pos}$ an extension. Then:*

1. *There is an injection from the set of predicate liftings of T' of arity p into the set of monotone predicate liftings of T of arity Vp.*
   In particular, the set of predicate liftings of T' of discrete arity n embeds into the monotone predicate liftings of T.
2. *In case T' is the posetification of T, the above mapping is a bijection.*

As a corollary, we obtain

Corollary 5.2. *Let T be a finitary Set-functor. If the posetification T' of T preserves embeddings, then the logic of all monotone predicate liftings of T is expressive.*

Remark 5.3. We know from [4] that if T preserves weak pullbacks then T' preserves embeddings. So the above theorem applies to weak-pullback preserving functors. This result was obtained in [18, Cor 6.9] already in a different way.

References

1. Abramsky, S.: A Cook's Tour of the Finitary Non-Well-Founded Sets. In: Artemov, S., Barringer, H., d'Avila Garcez, A., Lamb, L.C., Woods, J. (eds.) We Will Show Them: Essays in Honour of Dov Gabbay, vol. 1, pp. 1–18. College Publications (2005), `http://arxiv.org/pdf/1111.7148.pdf`
2. Adámek, J., Herrlich, J., Strecker, G.E.: Abstract and Concrete Categories. John Wiley & Sons (1990)
3. Adámek, J., Rosický, J.: Locally Presentable and Accessible Categories. London Math. Soc. LNS, vol. 189. Cambridge Univ. Press (1994)
4. Balan, A., Kurz, A.: Finitary Functors: From Set to Preord and Poset. In: Corradini, A., Klin, B., Cîrstea, C. (eds.) CALCO 2011. LNCS, vol. 6859, pp. 85–99. Springer, Heidelberg (2011)
5. Bílková, M., Kurz, A., Petrişan, D., Velebil, J.: Relation Liftings on Preorders and Posets. In: Corradini, A., Klin, B., Cîrstea, C. (eds.) CALCO 2011. LNCS, vol. 6859, pp. 115–129. Springer, Heidelberg (2011)

[6] Which can be identified with $\heartsuit \in \mathsf{Pos}(T'([p, \mathbb{2}]), \mathbb{2})$.

6. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge Tracts in Theor. Comput. Sci., vol. 53. Cambridge Univ. Press (2002)
7. Bonsangue, M.M., Kurz, A.: Presenting Functors by Operations and Equations. In: Aceto, L., Ingólfsdóttir, A. (eds.) FOSSACS 2006. LNCS, vol. 3921, pp. 172–186. Springer, Heidelberg (2006)
8. Bourke, J.: Codescent Objects in 2-Dimensional Universal Algebra, PhD Thesis, University of Sydney (2010)
9. Carboni, A., Street, R.: Order Ideals in Categories. Pacific J. Math. 124, 275–288 (1986)
10. Dunn, J.M.: Positive Modal Logic. Studia Logica 55, 301–317 (1995)
11. Guitart, R.: Rélations et Carrés Exacts. Ann. Sci. Math. Qué. 4, 103–125 (1980)
12. Johnstone, P.T.: Stone Spaces. Cambridge Univ. Press (1982)
13. Johnstone, P.T.: Vietoris Locales and Localic Semilattices. In: Hoffmann, R.E., Hofmann, K.H. (eds.) Continuous Lattices and their Applications. LNPAM, vol. 101, pp. 155–180. Marcel Dekker (1985)
14. Kelly, G.M.: Basic Concepts of Enriched Category Theory. London Math. Soc. LNS, vol. 64. Cambridge Univ. Press (1982), Reprint as: Theory Appl. Categ. 10 (2005)
15. Kelly, G.M.: Structures Defined by Finite Limits in the Enriched Context. I. Cah. Topol. Géom. Différ. Catég. 23, 3–42 (1982)
16. Kelly, G.M.: Elementary Observations on 2-Categorical Limits. Bull. Austral. Math. Soc. 39, 301–317 (1989)
17. Kelly, G.M., Lack, S.: Finite Product-Preserving-Functors, Kan Extensions and Strongly-Finitary 2-Monads. Appl. Categ. Struct. 1, 85–94 (1993)
18. Kurz, A., Leal, R.: Equational Coalgebraic Logic. ENTCS 249, 333–356 (2009)
19. Kurz, A., Rosický, J.: Strongly Complete Logics for Coalgebras, Logic. Meth. Comput. Sci. 8, 1–32 (2012)
20. Kurz, A., Velebil, J.: Enriched Logical Connections. Appl. Categ. Struct (2011) (online first)
21. Kurz, A., Velebil, J.: Quasivarieties and Varieties of Ordered Algebras: Regularity and exactness (February 2013), `http://www.cs.le.ac.uk/people/akurz/pos_ua.pdf` (submitted)
22. Lack, S., Rosický, J.: Notions of Lawvere Theory. Appl. Categ. Struct. 19, 363–391 (2011)
23. Mac Lane, S.: Categories for the Working Mathematician. GTM, vol. 5. Springer (1971)
24. Palmigiano, A.: A Coalgebraic View on Positive Modal Logic. Theor. Comput. Sci. 327, 175–195 (2004)
25. Pattinson, D.: Coalgebraic Modal Logic: Soundness, Completeness and Decidability of Local Consequence. Theor. Comput. Sci. 309, 177–193 (2003)
26. Robinson, E.: Powerdomains, Modalities and the Vietoris Monad. Comp. Lab. Tech. Report. 98, Univ. Cambridge (1986)
27. Schröder, L.: Expressivity of Coalgebraic Modal Logic: The Limits and Beyond. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 440–454. Springer, Heidelberg (2005)
28. Velebil, J., Kurz, A.: Equational Presentations of Functors and Monads. Math. Struct. Comput. Sci. 21, 363–381 (2011)
29. de Vink, E.P., Rutten, J.J.M.M.: Bisimulation for Probabilistic Transition Systems: A Coalgebraic Approach. Theor. Comput. Sci. 221, 271–293 (1999)
30. Winskel, G.: On Powerdomains and Modality. Theor. Comput. Sci. 36, 127–137 (1985)

Many-Valued Relation Lifting and Moss' Coalgebraic Logic

Marta Bílková[1,⋆] and Matěj Dostál[2,⋆⋆]

[1] Institute of Computer Science, Academy of Sciences of the Czech Republic
bilkova@cs.cas.cz
[2] Faculty of Electrical Engineering, Czech Technical University in Prague, Prague, Czech Republic
dostamat@math.feld.cvut.cz

Abstract. The notion of relation lifting can be generalised to work with many-valued relations while retaining many vital properties of the "classical" relation lifting. We show that polynomial endofunctors of the category of sets and mappings admit $\mathscr{V}$-relation lifting for relations taking values from a commutative quantale $\mathscr{V}$. Using the technique of functor presentations, we then show that every finitary weak pullback preserving functor admits a $\mathscr{V}$-relation lifting for $\mathscr{V}$ being a complete Heyting algebra. As an application of the many-valued lifting we inspect the notion of many-valued bisimulation and we introduce an expressive many-valued variant of Moss' logic for T-coalgebras, parametric in the functor T.

Keywords: coalgebra, coalgebraic logic, relation lifting, many-valued logic.

1 Introduction

Relation lifting [3], given a Set functor T, allows us to "lift" a relation $R : A \nrightarrow B$ to a relation $\overline{T}(R) : TA \nrightarrow TB$ in a functorial way. It is in particular useful to introduce the notion of bisimulation, and to define semantics for the cover modality ∇ in Moss' coalgebraic logic [13], [11].

In this paper we generalise the notion of relation lifting to many-valued relations, taking values from a certain many-valued structure $\mathscr{V}$. This generalisation will allow us to define a many-valued variant of Moss' coalgebraic logic.

Since the definition of relation lifting for *polynomial functors* is very intuitive, it is not hard to see that there are ways to define a generalised version of relation lifting even for many-valued relations. Showing that the intuitive inductive definition indeed works is the first result of this paper.

⋆ Marta Bílková acknowledges the support of the grant No. P202/10/1826 of the Czech Science Foundation.

⋆⋆ This work has been supported by grant no. SGS12/060/OHK3/1T/13 of SGS CVUT. The authors would also like to thank Jiří Velebil for invaluable help and useful comments, as well as to anonymous referees and their insightful comments.

R. Heckel and S. Milius (Eds.): CALCO 2013, LNCS 8089, pp. 66–79, 2013.

Proving that there is a many-valued variant of relation lifting for a larger class of *finitary* functors preserving weak pullbacks turns out to be harder: we use presentations of Set functors known from [2], and show how they can be used to define the standard, as well as the many-valued, relation lifting. To reach the goal, one needs to put some restrictions on the algebra $\mathscr{V}$. The final result is that there is a well-behaved $\mathscr{V}$-valued relation lifting for any weak pullback preserving finitary functor, given an arbitrary complete Heyting algebra $\mathscr{V}$.

The problem of many-valued relation lifting has already been discussed in the papers [4] and [5]. However, we take a different approach. We are interested in lifting endofunctors of the base category Set to the category $\mathscr{V}$-mat of sets and $\mathscr{V}$-valued relations, whereas the base category in [4] is the category of posets, or $\mathscr{V}$-categories in [5], and they both work essentially in an enriched setting. Thus the question we have asked ourselves at the beginning was: how far does one get in defining many-valued relation lifting while staying in the category Set?

Having a working notion of $\mathscr{V}$-valued relation lifting at hand, we turn our attention to its applications. First we study a notion of $\mathscr{V}$-bisimulation which is a direct generalisation of the standard, two-valued notion of bisimulation, and the resulting notion of $\mathscr{V}$-bisimilarity, defined as existence of a $\mathscr{V}$-bisimulation. We conclude that, as expected, $\mathscr{V}$-bisimilarity coincides with the two-valued bisimilarity and therefore captures behavioural equivalence. Then we consider Moss' coalgebraic logic [13] and study its many-valued variant: given a functor T, for which the lifting exists, we introduce a many-valued cover modality ∇ of arity T, and define its semantics using the $\mathscr{V}$-lifting of the many-valued satisfaction relation $\Vdash$.

The semantics of many-valued Moss' logic, defined by the many-valued relation lifting, allows us to prove that the logic is adequate and expressive with respect to bisimilarity. To be able to show this we need to restrict the algebra $\mathscr{V}$ once more, namely we prove that the introduced logic is adequate and expressive for $\mathscr{V}$ being a Gödel chain.

The preliminary Section 2 is meant mainly for fixing the notation and it introduces the concepts we work with in a way which directly admits generalisations. In Section 3 we define the notion of many-valued relation lifting and show how it retains the properties of the classical relation lifting. Section 4 then deals with the notions definable by relation lifting: the $\mathscr{V}$-bisimulation and $\mathscr{V}$-Moss' logic with many-valued semantics.

2 Preliminaries

The following definition introduces algebraic structures we use to model values in the many-valued setting.

Definition 1. *A* commutative quantale $\mathscr{V}$ *is a complete lattice* $(V, \bigwedge, \bigvee)$ *with the structure of a commutative monoid* $(V, \otimes, \mathsf{e})$ *such that the tensor distributes over arbitrary joins:*

$$\left(\bigvee_i x_i\right) \otimes y = \bigvee_i (x_i \otimes y). \tag{1}$$

A commutative quantale $\mathscr{V}$ with $\otimes = \wedge$ and $\mathsf{e} = \top$ is called a complete Heyting algebra. *A complete Heyting algebra $\mathscr{V}$ is a* Gödel chain *if the ordering relation $\leq$ of the underlying lattice of $\mathscr{V}$ is a linear order.*

Definition 2 (Many-valued Relations). *For a given quantale $\mathscr{V}$, a $\mathscr{V}$-*relation *$R : A \nrightarrow B$ is a function $R : A \times B \to V$. The composition of two $\mathscr{V}$-relations $R : A \nrightarrow B$ and $S : B \nrightarrow C$ is a relation $S \circ R : A \nrightarrow C$ such that*

$$(S \circ R)(a, c) = \bigvee_{b \in B} (R(a, b) \otimes S(b, c)) \tag{2}$$

holds in $\mathscr{V}$. We can form an opposite $\mathscr{V}$-relation $R^{op} : B \nrightarrow A$ and given subsets $A' \subseteq A$ and $B' \subseteq B$, we can restrict R to $R \restriction_{A' \nrightarrow B'}$ in the same way as with two-valued relations. Given moreover a $\mathscr{V}$-relation $S : A \nrightarrow B$, we say that $R \leq S$ if and only if $R(a, b) \leq S(a, b)$ for all $a \in A$ and $b \in B$.

Sets and $\mathscr{V}$-relations together with $\mathscr{V}$-relation composition form a category $\mathscr{V}$-mat, where $\mathscr{V}$-mat stands for $\mathscr{V}$-matrices. Observe that the composition formula is a generalisation of matrix multiplication (which can be used to compute the usual relation composition as well).

Example 3 (Various Functors). We will work with a plenitude of Set functors. Recall that $T : \mathsf{Rel} \to \mathsf{Rel}$ is a *2-functor* if $T(R) \leq T(S)$ holds for $R \leq S$, and it is a *lax functor* provided the inequality $T(S) \circ T(R) \leq T(S \circ R)$ holds instead of the full equality.

- Polynomial functors are defined by the following Backus-Naur form:

$$T ::= C \mid \mathsf{Id} \mid T \times T \mid \coprod_i T_i. \tag{3}$$

- The powerset functor is denoted by $\mathcal{P}$, and its finitary coreflection by $\mathcal{P}_\omega$. For any set X we have $\mathcal{P}_\omega(X) = \{Y \mid Y \subseteq X, Y \text{ finite}\}$.
- The graph functor $\mathsf{gr} : \mathsf{Set} \to \mathsf{Rel}$ is the inclusion of functions into relations.
- The relation inclusion functor $\mathcal{I} : \mathsf{Rel} \to \mathscr{V}$-$\mathsf{mat}$ maps a relation $R : A \nrightarrow B$ into a $\mathscr{V}$-relation $\mathcal{I}(R) : A \nrightarrow B$. We have $\mathcal{I}(R)(a, b) = \mathsf{e}$ if $R(a, b)$ holds, otherwise $\mathcal{I}(R)(a, b) = \bot$.
- The $\mathscr{V}$-graph functor $\mathsf{Gr} : \mathsf{Set} \to \mathscr{V}$-$\mathsf{mat}$ is the composition $\mathcal{I} \circ \mathsf{gr}$.
- Let $v \in \mathscr{V}$ be an element of a Gödel chain. The cut lax functor $(-)^{\geq v} : \mathscr{V}$-$\mathsf{mat} \to \mathsf{Rel}$ is defined as follows: $R^{\geq v}(a, b)$ if and only if $R(a, b) \geq v$. Similarly, $(-)^{>v} : \mathscr{V}$-$\mathsf{mat} \to \mathsf{Rel}$ is an honest functor.

As an important example we introduce another functor, which is a many-valued generalisation of the powerset functor.

Example 4 ($\mathscr{V}$-powerset Functor). Let $\mathscr{V}$ be a complete Heyting algebra. We define $\mathcal{P}_\mathcal{V}$ on objects as $\mathcal{P}_\mathcal{V}(A) = V^A$. Given a function $f : A \to B$ the "direct image" function $\mathcal{P}_\mathcal{V}(f) : V^A \to V^B$ maps $\alpha : A \to V$ to the function $\beta : B \to V$, where $\beta(b) = \bigvee_{\{a \in A \mid f(a) = b\}} \alpha(a)$.

This functor preserves weak pullbacks. We shall deal exclusively with the finitary coreflection of this functor: this means that $\mathcal{P}_{\mathcal{V}}(A)$ will be only the functions from V^A which have finite support.

Remark 5 (Base). Given an element $\alpha \in TA$, we define $\mathsf{Base}^T_A(\alpha)$ to be the smallest finite set $S \subseteq A$ such that $\alpha \in TS$, see [11]. For a set A, we thus obtain a mapping $\mathsf{Base}^T_A : TA \to \mathcal{P}_\omega A$, giving rise to a natural transformation $\mathsf{Base}^T : T \to \mathcal{P}_\omega$, provided T is finitary, standard and preserves weak pullbacks (for a functor to admit a transformation to the powerset monad, the finitarity can be dropped and the preservation property can be weakened: it has been shown in [10] that when T weakly preserves preimages and intersections, the base exists and is natural).

Finitary functors can be presented by means of polynomial functors. We will use this fact when defining many-valued relation lifting for finitary functors in Section 3.

Theorem 6 (Functor Presentations [2]). *Every finitary endofunctor of* Set *is* presentable *by a* canonical *presentation. This means that there is a functor*

$$H_\Sigma = \coprod_{n\in\mathbb{N}} \mathsf{Id}^n \times T(n)$$

together with a natural epimorphism $\varepsilon : H_\Sigma \to T$.

Note 7. We denote the canonical presentation of T by (H_Σ, ε). The elements of $H_\Sigma A$ are pairs $(\mathbf{a} : n \to A, \sigma \in Tn)$. We denote this pair as a *term* $\sigma(\mathbf{a})$, as σ is an operation and $\mathbf{a}$ a vector of elements from A. If $\varepsilon_A(\sigma(\mathbf{a})) = \alpha$, the term $\sigma(\mathbf{a})$ *represents* α. A generic term representing α is denoted by t_α. In general, we can replace the set $T(n)$ in the definition with an arbitrary set for each n. This gives a presentation which is in a sense a subpresentation of the canonical presentation, see [12].

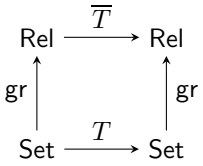
Example 8 (Some Presentations). In the canonical presentation for the functor $\mathcal{P}_\omega$, the n-ary operations are elements $\sigma \in \mathcal{P}_\omega(n)$, and for a set A and a vector $\mathbf{a} : n \to A$ the term $\sigma(\mathbf{a})$ represents the subset $\{\mathbf{a}(m) \mid m \in \sigma\} \in \mathcal{P}_\omega(A)$.

The canonical presentation for the functor $\mathcal{P}_{\mathcal{V}}$ has the elements $\sigma \in \mathcal{P}_{\mathcal{V}}(n)$ as n-ary operations: these are functions $\sigma : n \to V$. Together with a vector $\mathbf{a} : n \to A$, the term $\sigma(\mathbf{a})$ represents the function $\alpha : A \to V$ such that $\alpha(a) = \bigvee_{\mathbf{a}(m)=a} \sigma(m)$.

Weak pullback preserving functors enjoy the property of having *dominated* presentations. This property will become crucial in the proof of functoriality of our definition of the many-valued lifting (Definition 19), we will define dominated presentations in the proof of Theorem 14.

In the rest of this section, we state some basic facts about relation lifting. The following theorem of Trnková [14] characterizes functors admitting a functorial lifting from the category of sets to the category of sets and relations. The result can be also derived from Barr's paper [3].

Theorem 9 (Lifting Theorem — [14,3]). *A set functor T preserves weak pullbacks if and only if there exists a 2-functor $\overline{T} : \mathsf{Rel} \to \mathsf{Rel}$ such that the square on the right commutes:*



The relation lifting resulting from the theorem can be explicitely computed as follows:

Definition 10 (Relation Lifting). *Given a relation $R : A \nrightarrow B$ and a functor T, the* relation lifting $\overline{T}R$ *of the relation R is defined as follows:*

$$\overline{T}R = \{(\alpha, \beta) \mid (\exists \gamma \in TR) : Tp_1(\gamma) = \alpha \;\; \textit{and} \;\; Tp_2(\gamma) = \beta\}.$$

Example 11 ((Many-valued) Powerset Lifting). Two elements $\alpha \in \mathcal{P}_\omega(A)$, $\beta \in \mathcal{P}_\omega(B)$ are in the lifted relation $\overline{\mathcal{P}_\omega}R$ if and only if the following conditions (the Egli-Milner lifting) hold:

$$\forall a \in \alpha \; \exists b \in \beta : \; R(a,b), \quad \forall b \in \beta \; \exists a \in \alpha : \; R(a,b). \tag{4}$$

In the case of the functor $\mathcal{P}_\mathcal{V}$, we get that $\overline{\mathcal{P}_\mathcal{V}}(R)(\alpha, \beta)$ holds if and only if

$$\forall a \in A : \bigvee_{R(a,b)} \beta(b) = \alpha(a), \quad \forall b \in B : \bigvee_{R(a,b)} \alpha(a) = \beta(b). \tag{5}$$

If $\mathcal{V}$ is a chain, this simplifies to

$$\forall a \in A, \alpha(a) > \bot : \; \exists b \in B : (R(a,b) \wedge (\alpha(a) \leq \beta(b))), \tag{6}$$
$$\forall b \in B, \beta(b) > \bot : \; \exists a \in A : (R(a,b) \wedge (\beta(b) \leq \alpha(a))). \tag{7}$$

Relation lifting can also be computed using presentations of finitary functors. The following definition of lifting using presentations coincides with the one given above.

Lemma 12 (Relation Lifting via Presentations [12]). *For a relation $R : A \nrightarrow B$ and a finitary* Set *endofunctor T with its presentation (H_Σ, ε), the following two properties are equivalent:*

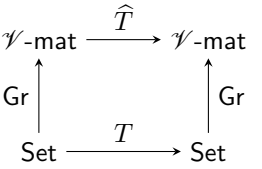
- *$\overline{T}(R)(\alpha, \beta)$ holds for $\alpha \in T(A)$, $\beta \in T(B)$.*
- *There exist $t_\alpha \in HA$ representing α and $t_\beta \in HB$ representing β such that $\overline{H}(R)(t_\alpha, t_\beta)$.*

Remark 13. The standard, two-valued, relation lifting allows us to define bisimulations in a succinct way: we say that for two given coalgebras $c : A \to TA$, $d : B \to TB$, a relation $R : A \nrightarrow B$ is a bisimulation if for any pair $R(a,b)$ it follows that $\overline{T}(B)(c(a), d(b))$. We say that the states a and b related by R are bisimilar. In the case that T preserves weak pullbacks, bisimilarity precisely captures behavioural equivalence.

3 Many-Valued Relation Lifting

In this section we introduce many-valued relation lifting. Subsequently, this lifting will be used to define a many-valued variant of Moss' coalgebraic logic. We also study the notion of many-valued bisimulation which stems from this approach. We will first introduce many-valued relation lifting for polynomial functors via operations on many-valued relations. Then, using presentations of functors, we will extend this notion of lifting to finitary weak pullback preserving functors. This extension will force us to work with complete Heyting algebras instead of general quantales.

Theorem 14 (Many-valued Relation Lifting). *For a finitary* Set *endofunctor T preserving weak pullbacks, there exists a 2-functor $\widehat{T}$ on the category $\mathscr{V}$-*mat *which makes the diagram on the right commute:*



We say that such a functor $\widehat{T}$ is a $\mathscr{V}$-lifting of T. The rest of this section is dedicated to finding a $\mathscr{V}$-lifting generalising Definition 10 and showing that $\mathscr{V}$-lifting retains many of the properties of the "standard" relation lifting.

We will define $\mathscr{V}$-lifting for polynomial functors as a straightforward generalisation of the standard relation lifting using two operations on $\mathscr{V}$-relations.

Definition 15. *For two $\mathscr{V}$-relations $R_1 : A_1 \nrightarrow B_1$ and $R_2 : A_2 \nrightarrow B_2$, we define their* sum *to be a relation $R_1 \boxplus R_2 : A_1 + A_2 \nrightarrow B_1 + B_2$ defined as follows:*

$$R_1 \boxplus R_2(\alpha, \beta) = \begin{cases} R_i(\alpha, \beta) & \text{if } \alpha \in A_i \text{ and } \beta \in B_i \\ \bot & \text{otherwise.} \end{cases} \tag{8}$$

The tensor *of R_1 and R_2 is a relation $R_1 \boxtimes R_2 : A_1 \times A_2 \nrightarrow B_1 \times B_2$ defined on elements as follows:*

$$R_1 \boxtimes R_2((a_1, a_2), (b_1, b_2)) = R_1(a_1, b_1) \otimes R_2(a_2, b_2). \tag{9}$$

Definition 16 ($\mathscr{V}$-lifting for Polynomial Functors). *We define the $\mathscr{V}$-lifting $\widehat{T} : \mathscr{V}$-mat $\to \mathscr{V}$-mat for a polynomial endofunctor T of* Set *inductively:*

- *For the constant functor $T = C$, we define $\widehat{T}$ as the constant functor $C : \mathscr{V}$-mat $\to \mathscr{V}$-mat.*
- *For the identity functor $T = \mathsf{Id}$, we define $\widehat{T}$ as the identity functor on $\mathscr{V}$-mat.*
- *For the sum $T = T_1 + T_2$ of functors T_1 and T_2, we define $\widehat{T}$ by its action on morphisms as follows: $\widehat{T}(R) = \widehat{T_1}(R) \boxplus \widehat{T_2}(R)$.*
- *For the product $T = T_1 \times T_2$ of functors T_1 and T_2, we define $\widehat{T}$ by its action on morphisms as follows: $\widehat{T}(R) = \widehat{T_1}(R) \boxtimes \widehat{T_2}(R)$*

The following theorem shows that this definition is correct:

Theorem 17. *Definition 16 yields a $\mathscr{V}$-lifting in the sense of Theorem 14. It satisfies the following requirements:*

- *It constitutes a 2-functor* $\widehat{T}$.
- *It is an extension of standard relation lifting:* $\mathcal{I} \circ \overline{T} = \widehat{T} \circ \mathcal{I}$.
- *Since* $\widehat{T}$ *extends the standard lifting, it commutes with the graph functor* Gr.

Proof (Theorem 17). To prove that $\widehat{T}$ is a functor for a polynomial T is immediate for the case of a constant functor C and identity functor Id. The other two cases follow from the following distributive law for sums:

$$(M \circ N) \boxplus (O \circ P) = (M \boxplus O) \circ (N \boxplus P), \tag{10}$$

and a similar one for products. Proving that $\widehat{T}$ is a 2-functor is a straightforward computation. The fact that $\mathscr{V}$-lifting extends standard lifting comes from the observation that setting $\mathscr{V} = 2$ yields the standard definition of relation lifting. The proof that $\mathscr{V}$-lifting commutes with the graph functor Gr then comes from the fact that $\mathsf{gr} \circ T = \overline{T} \circ \mathsf{gr}$ and $\mathcal{I} \circ \overline{T} = \widehat{T} \circ \mathcal{I}$ holds. □

Example 18 (The Polynomial Functor $Z \times \mathsf{Id}$*).* Given a $\mathscr{V}$-relation $R : A \nrightarrow B$ and a functor $Z \times \mathsf{Id}$, the lifted $\mathscr{V}$-relation $\widehat{Z \times \mathsf{Id}}(R) : Z \times A \nrightarrow Z \times B$ is defined by the following formula:

$$\widehat{Z \times \mathsf{Id}}(R)((z_1, a)(z_2, b)) = \begin{cases} R(a,b) & \text{if } z_1 = z_2 \\ \bot & \text{otherwise.} \end{cases} \tag{11}$$

The property of standard relation lifting stated in Lemma 12 hints for a generalisation to the $\mathscr{V}$-valued case, suggesting a definition of lifting for all finitary functors preserving weak pullbacks. From now on we restrict ourselves to work with a complete Heyting algebra $\mathscr{V}$.

Definition 19 ($\mathscr{V}$-lifting of Finitary Functors). *Given a* $\mathscr{V}$*-relation* $R : A \nrightarrow B$ *and a finitary weak pullback preserving functor* T*, we define the lifted* $\mathscr{V}$*-relation* $\widehat{T}R : TA \nrightarrow TB$ *by the following formula for* $\alpha \in TA$ *and* $\beta \in TB$ *using the canonical presentation* (H_Σ, ε)*:*

$$\widehat{T}(R)(\alpha, \beta) = \bigvee_{t_\alpha, t_\beta} \widehat{H_\Sigma}(R)(t_\alpha, t_\beta).$$

The notation in the previous definition is slightly relaxed: we take the supremum over all pairs (t_α, t_β) such that they represent α and β respectively.

Remark 20. By the above definition we have not redefined $\mathscr{V}$-lifting for polynomial functors: even if the canonical presentation of a polynomial functor need not be the functor itself, it is not hard to see that the two definitions coincide.

The following example for the powerset functor $\mathcal{P}_\omega$ shows that the restriction to complete Heyting algebras is necessary.

Example 21. Consider a unit interval with multiplication $\mathscr{V} = ([0,1], \cdot, 1)$ as a quantale and $A = \{a\}, B = \{b\}, C = \{c_1, c_2\}$. Set two relations $R : A \nrightarrow B$, $S : B \nrightarrow C$ as follows: $R(a,b) = \frac{1}{2}$, $S(b,c_1) = \frac{1}{3}$, $S(b,c_2) = \frac{1}{5}$. These data show that $\widehat{\mathcal{P}_\omega}$ is not a functor. We have $\widehat{\mathcal{P}_\omega}(S \circ R)(\{a\}, \{c_1, c_2\}) = \frac{1}{60}$, but $\widehat{\mathcal{P}_\omega}(S) \circ \widehat{\mathcal{P}_\omega}(R)(\{a\}, \{c_1, c_2\}) = \frac{1}{30}$.

Now we are ready to prove Theorem 14 in its generality.

Proof (Proof of Theorem 14). We show the harder part of the proof only. For any two $\mathscr{V}$-relations $R : A \nrightarrow B$ and $S : B \nrightarrow C$ we need to prove the equality

$$\widehat{T}(S \circ R) = \widehat{T}(S) \circ \widehat{T}(R).$$

To prove that for any $\alpha \in TA$ and $\gamma \in TC$ the inequality

$$\widehat{T}(S \circ R)(\alpha, \gamma) \leq \widehat{T}(S) \circ \widehat{T}(R)(\alpha, \gamma) \tag{12}$$

holds is easy: it suffices to find for for any triple t_α, s, t_γ a quadruple $t_\alpha, t_\beta, t'_\beta, t_\gamma$ witnessing the inequality (12). Denote by β the element represented by s. Then we can set $t_\beta = t'_\beta = s$, and this is a witness for the inequality.

For the other inequality it would be enough to find for every quadruple $t_\alpha, t_\beta, t'_\beta, t_\gamma$ a triple t'_α, s, t'_γ such that

$$\widehat{H_\Sigma} R(t_\alpha, t_\beta) \otimes \widehat{H_\Sigma} S(t'_\beta, t_\gamma) \leq \widehat{H_\Sigma} R(t'_\alpha, s) \otimes \widehat{H_\Sigma} S(s, t'_\gamma) \tag{13}$$

Here we use the fact that weak pullback preserving functors have dominated canonical presentations. A presentation (H_Σ, ε) of a functor T is called *dominated* (see [1]) if for any two terms $\sigma(\mathbf{x})$, $\tau(\mathbf{y})$ representing $\alpha \in T(A)$ (where $\mathbf{x} : n \to A$, $\mathbf{y} : m \to A$ and $\sigma \in T(n)$, $\tau \in T(m)$) there is an operation $\rho \in T(k)$ together with two maps $u : k \to n$ and $v : k \to m$ such that $T(u)(\rho) = \sigma$, $T(v)(\rho) = \tau$ and $\mathbf{x} \circ u = \mathbf{y} \circ v$.

Let $t_\alpha = \sigma(\mathbf{a})$, $t_\gamma = \tau(\mathbf{c})$ for some $\sigma \in T(n)$, $\tau \in T(m)$. We can assume for the other two terms t_β and t'_β from the quadruple that they are of the form $t_\beta = \sigma(\mathbf{x})$ and $t'_\beta = \tau(\mathbf{y})$

Since both t_β and t'_β represent β and since (H_Σ, ε) is a dominated presentation, there is an operation $\rho \in T(k)$ together with two maps $u : k \to n$ and $v : k \to m$ such that $\mathbf{x} \circ u = \mathbf{y} \circ v$ and the operation ρ gets mapped to σ and τ by $T(u)$ and $T(v)$. Therefore, $\rho(\mathbf{x} \circ u)$ represents β. Set $s = \rho(\mathbf{x} \circ u)$. We now just have to find suitable terms t'_α and t'_γ which would witness the inequality (13). The terms t'_α and t'_γ are now forced to have ρ as the operation, since $s = \rho(\mathbf{x} \circ u)$.

Let $t'_\alpha = \rho(\mathbf{a} \circ u)$ and $t'_\gamma = \rho(\mathbf{c} \circ v)$. These terms represent α and γ respectively. A tedious but straightforward computation shows that these terms are indeed the desired witnesses.

The functor $\widehat{T}$ is actually a 2-functor. For a finitary functor T we use its canonical presentation (H_Σ, ε), where H_Σ is a polynomial functor. Given two relations $R : A \nrightarrow B$ and $S : A \nrightarrow B$ with $R \leq S$, we see that

$$\widehat{T}R(\alpha, \beta) = \bigvee_{t_\alpha, t_\beta} \widehat{H_\Sigma} R(t_\alpha, t_\beta) \leq \bigvee_{t_\alpha, t_\beta} \widehat{H_\Sigma} S(t_\alpha, t_\beta) = \widehat{T}S(\alpha, \beta).$$

□

The $\mathscr{V}$-lifting has some nice properties that can be proved easily.

Fact 22. *$\mathscr{V}$-lifting of finitary functors enjoys the following properties:*

– *It commutes with opposites:* $\widehat{T}(R^{op}) = \widehat{T}(R)^{op}$.
– *It can be computed by restricting to bases:*

$$\widehat{T}(R)(\alpha,\beta) = \widehat{T}(R \restriction_{\mathsf{Base}(\alpha)\nrightarrow\mathsf{Base}(\beta)})(\alpha,\beta). \tag{14}$$

– *It can be computed by cuts:*

$$\widehat{T}(R)(\alpha,\beta) = \bigvee_{v\in V} v \wedge \mathcal{I}(\overline{T}(R^{\geq v}))(\alpha,\beta). \tag{15}$$

Remark 23. The fact that $\mathscr{V}$-lifting can be computed by cuts implies that $\mathscr{V}$-lifting does not depend on the choice of presentation, provided it is dominated.

To show examples $\mathscr{V}$-liftings for more involved functors, we restrict ourselves to a Gödel chain $\mathscr{V}$.

Example 24 (Finitary Powerset). Let $R : A \nrightarrow B$ be a $\mathscr{V}$-relation and $\alpha \in \mathcal{P}_\omega A$, $\beta \in \mathcal{P}_\omega B$ two finite subsets. Their $\mathscr{V}$-lifting is given by the following equation

$$\widehat{\mathcal{P}_\omega}(R)(\alpha,\beta) = \left(\bigwedge_{a\in\alpha}\bigvee_{b\in\beta} R(a,b)\right) \wedge \left(\bigwedge_{b\in\beta}\bigvee_{a\in\alpha} R(a,b)\right), \tag{16}$$

which is a direct generalisation of the Egli-Milner lifting.

Example 25 (Many-valued Powerset). Taking again a $\mathscr{V}$-relation with $\alpha \in \mathcal{P}_\mathcal{V} A$, $\beta \in \mathcal{P}_\mathcal{V} B$ two finite "fuzzy" subsets, their lifting is computed as follows

$$\widehat{\mathcal{P}_\mathcal{V}}(R)(\alpha,\beta) = \left(\bigwedge_{a\in\mathsf{supp}(\alpha)}\bigvee_{\substack{b\in\mathsf{supp}(\beta)\\ \alpha(a)\leq\beta(b)}} R(a,b)\right) \wedge \left(\bigwedge_{b\in\mathsf{supp}(\beta)}\bigvee_{\substack{a\in\mathsf{supp}(\alpha)\\ \beta(b)\leq\alpha(a)}} R(a,b)\right) \tag{17}$$

where $\mathsf{supp}(\alpha)$ marks the support of α. Note that again the shape of the formula resembles the Egli-Milner lifting.

4 A Many-Valued Coalgebraic Logic

Having a working notion of $\mathscr{V}$-lifting at hand, we can use it to define the semantics of a many-valued variant of finitary Moss' coalgebraic logic, and show examples of the logic for coalgebras of various Set functors. Moss' coalgebraic logic originated in Moss' paper [13], and its finitary variant extending the boolean logic has been explored by various authors, see e.g. [11] for an extensive exposure of the logic, including an axiomatization and a completeness proof.

Using the notion of $\mathscr{V}$-lifting, we can also define a notion of $\mathscr{V}$-bisimulation. We show that, in the case that $\mathscr{V}$ is a Gödel chain, the many-valued Moss' logic is expressive for the resulting notion of bisimilarity.

4.1 Logic and Examples

Let us fix a finitary functor T and a complete Heyting algebra $\mathscr{V}$ with the operations $\wedge$, $\vee$ and $\to$ (the Heyting implication, i.e. the residuum of $\wedge$). We are going to introduce the syntax and semantics of Moss' many-valued logic ($\mathscr{V}$-logic), which very much resembles the standard definition of Moss' logic. For the sake of expressivity, we include the elements of the Heyting algebra into the syntax of our language as canonical constants. The modal part of the language consists in the single operator ∇, which has the coalgebra functor T as its arity. It is sometimes called the *cover* modality.

Definition 26 (Syntax of Moss' $\mathscr{V}$-logic). *With a fixed set of atomic propositions* At*, we construct the language* $\mathcal{L}$ *of the coalgebraic logic inductively:*

$$\varphi ::= v \mid s \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \to \varphi \mid \nabla\alpha, \tag{18}$$

where $v \in V$*,* $s \in \mathsf{At}$ *and* $\alpha \in T\mathcal{L}$*.*

The definition of semantics uses the operations of the Heyting algebra $\mathscr{V}$.

Definition 27 (Semantics of Moss' $\mathscr{V}$-logic). *For a coalgebra* $c : A \to TA$ *together with the atomic evaluation function* $\mathsf{ev}_a : \mathsf{At} \to V$ *for every state* $a \in A$*, we define the satisfaction* $\mathscr{V}$*-relation* $\Vdash_c \colon A \nrightarrow \mathcal{L}$ *inductively as follows:*

$$\begin{aligned}
a \Vdash_c v &= v,\\
a \Vdash_c s &= \mathsf{ev}_a(s),\\
a \Vdash_c \varphi \wedge \psi &= (a \Vdash_c \varphi) \wedge (a \Vdash_c \psi),\\
a \Vdash_c \varphi \vee \psi &= (a \Vdash_c \varphi) \vee (a \Vdash_c \psi),\\
a \Vdash_c \varphi \to \psi &= (a \Vdash_c \varphi) \to (a \Vdash_c \psi),\\
a \Vdash_c \nabla\alpha &= c(a)\, \widehat{T}(\Vdash_c)\, \alpha,
\end{aligned}$$

where again $v \in V$ *and* $s \in \mathsf{At}$*.*

Now we will show examples of $\mathscr{V}$-logics for functors whose $\mathscr{V}$-lifting we have computed in Section 3.

Example 28 (Logic for Streams). For a coalgebra $c : A \to Z \times A$ for the stream functor $Z \times \mathsf{Id}$ and a state $a \in A$, we shall look at the meaning of the modal formula $\nabla(z, \varphi)$. Let $c(a) = (z', a')$. Then

$$a \Vdash_c \nabla(z, \varphi) = \begin{cases} a' \Vdash_c \varphi & \text{if } z = z' \\ \bot & \text{otherwise.} \end{cases}$$

Example 29 (Logic for the Functor $\mathcal{P}_\omega$). Given a coalgebra $c : A \to \mathcal{P}_\omega A$ for the powerset functor, a finite set α of formulas from $\mathcal{L}$, and considering that $\mathscr{V}$ is a Gödel chain, the semantics of the modal formula $\nabla(\alpha)$ is defined as follows, compare to Example 24:

$$a \Vdash_c \nabla\alpha = \left(\bigwedge_{a' \in c(a)} \bigvee_{\varphi \in \alpha} a' \Vdash_c \varphi \right) \wedge \left(\bigwedge_{\varphi \in \alpha} \bigvee_{a' \in c(a)} a' \Vdash_c \varphi \right).$$

Remark 30. Observe that $\mathcal{P}_\omega$-coalgebras are crisp Kripke frames considered by Bou et al. in [6] (and introduced by Fitting in [8,9]): we can understand such a colagebra as a set of states equipped with a two-valued accessibility relation, plus a many-valued valuation. We restrict ourselves to image-finite frames. The many-valued box and diamond modalities used in [6] are interdefinable with the ∇ modality presented here.

$$\Diamond\varphi \equiv \nabla\{\varphi, \top\}, \quad \Box\varphi \equiv \nabla\{\varphi\} \vee \nabla\emptyset.$$

Example 31 (Logic for the Functor $\mathcal{P}_\mathcal{V}$). Coalgebras for the functor $\mathcal{P}_\mathcal{V}$ are again the same structures as those introduced by Fitting in [8,9] as Heyting valued frames, and considered by Bou et al. in [6] as many-valued Kripke frames: namely, we can understand such a coalgebra as a set of states equipped with a many-valued accessibility relation, plus a many-valued valuation. Consider a coalgebra $c : A \to \mathcal{P}_\mathcal{V} A$ and consider a Gödel chain $\mathscr{V}$. Then the semantics of ∇ is given by the following formula, compare to Example 25:

$$a \Vdash_c \nabla\alpha = \left(\bigwedge_{a' \in \mathsf{supp}(c(a))} \bigvee_{\substack{\varphi \in \mathsf{supp}(\alpha) \\ c(a)(a') \le \alpha(\varphi)}} a' \Vdash_c \varphi \right) \wedge \left(\bigwedge_{\varphi \in \mathsf{supp}(\alpha)} \bigvee_{\substack{a' \in \mathsf{supp}(c(a)) \\ \alpha(\varphi) \le c(a)(a')}} a' \Vdash_c \varphi \right).$$

4.2 Bisimulation and Expressivity

In this section we will look at the notion of $\mathscr{V}$-bisimulation which can be defined using $\mathscr{V}$-lifting, as a straightforward generalisation of the notion of standard bisimulation mentioned in Remark 13.

Definition 32 ($\mathscr{V}$-Bisimulation and $\mathscr{V}$-Bisimilarity). *A $\mathscr{V}$-relation $R : A \nrightarrow B$ is a* $\mathscr{V}$-bisimulation *for coalgebras $c : A \to TA$ and $d : B \to TB$ if for every $a \in A$ and $b \in B$*

$$R(a, b) \le \widehat{T}(R)(c(a), d(b)) \tag{19}$$

holds. States a and b are called $\mathscr{V}$-bisimilar *if there exists a $\mathscr{V}$-bisimulation R such that $R(a, b) > \bot$.*

Given furthermore an atomic evaluation functions ev *for the states, the relation $M : A \nrightarrow B$ defined as*

$$M(a, b) = \begin{cases} \mathsf{e} & \text{if } \mathsf{ev}_a = \mathsf{ev}_b \\ \bot & \text{otherwise.} \end{cases} \tag{20}$$

is called atomic harmony. *A $\mathscr{V}$-bisimulation $R : A \nrightarrow B$ with the property $R \le M$ is called* $\mathscr{V}$-bisimulation with atomic harmony.

Remark 33. From the properties of the $\mathscr{V}$-relation lifting, namely that it commutes with opposites and is functorial, it follows that $\mathscr{V}$-bisimilarity is reflexive, transitive and symmetric, i.e. it is an equivalence relation. While bisimulations

are many-valued, the bisimilarity notion we adopt is two-valued. It doesn't come as a surprise that it in fact coincides with the standard notion of bisimilarity, and consequently captures behavioural equivalence.

Fact 34 (Bisimilarity Coincides With $\mathscr{V}$-bisimilarity). *A standard bisimulation can be interpreted as a $\mathscr{V}$-bisimulation. Therefore, if two states are bisimilar, they are $\mathscr{V}$-bisimilar as well.*

For any $\mathscr{V}$-bisimulation R, its cut $R^{>\bot}$ is a bisimulation. Therefore, if two states are $\mathscr{V}$-bisimilar, they are bisimilar as well.

Next we will prove that the many-valued Moss' logic has the Hennessy-Milner property — it is adequate and expressive with respect to the notion of bisimilarity introduced above, provided $\mathscr{V}$ is a Gödel chain.

Theorem 35 (Expressivity of $\mathscr{V}$-logic). *Let $\mathscr{V}$ be a Gödel chain and $c : A \to TA$ and $d : B \to TB$ coalgebras.*

1. *(Adequacy.) Let $R : A \nrightarrow B$ be a $\mathscr{V}$-bisimulation with atomic harmony. Then for all $\varphi \in \mathcal{L}$ the following inequality holds:*

$$(R(a,b) \wedge a \Vdash_c \varphi) \leq b \Vdash_d \varphi.$$

2. *(Expressivity.) The relation $R : A \nrightarrow B$ defined as*

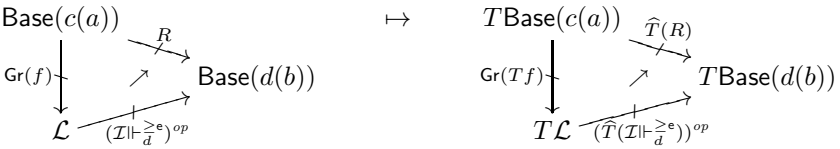
$$R(a,b) \;\; \textit{iff for all } \varphi \in \mathcal{L} : a \Vdash_c \varphi = b \Vdash_d \varphi$$

is a bisimulation with atomic harmony.

Proof. We only sketch the proof of expressivity. We show, that $R(a,b)$ entails $\widehat{T}(R)(c(a),d(b)) = \mathsf{e}$, proving that R is a bisimulation. For each $a_i \in \mathsf{Base}(c(a))$ and $b_j \in \mathsf{Base}(d(b))$, such that $R(a_i,b_j) \neq \mathsf{e}$, we fix a single formula ϕ_{ij} such that $a \Vdash_c \phi_{ij} \neq b \Vdash_d \phi_{ij}$. Then we define a function $f : \mathsf{Base}(a) \to \mathcal{L}$ as follows (using the presence of the canonical constants in the language):

$$f(a) = \bigwedge_{\{b_j \mid R(a_i,b_j) \neq \mathsf{e}\}} (\phi_{ij} \leftrightarrow (a \Vdash_c \phi_{ij})).$$

Then $a_i \Vdash_c f(a_i) = \mathsf{e}$, and $b_j \Vdash_d f(a_i) = \mathsf{e}$ entails $R(a_i,b_j) = e$. The former means that $\mathsf{Gr}(f) \leq \Vdash_c$, therefore $\mathsf{Gr}(Tf) \leq \widehat{T}(\Vdash_c)$. Putting $\alpha = (Tf)c(a)$ we know that $c(a)\ \widehat{T}(\Vdash_c)\ \alpha = \mathsf{e}$ and therefore $a \Vdash_c \nabla\alpha = \mathsf{e}$. From the latter we obtain the following lax commutative diagram in $\mathscr{V}$-mat, and, using moreover the properties of the lifting, its lax commutative lifting:



Since $a \Vdash_c \nabla\alpha = \mathsf{e}$ and $R(a,b)$, also $b \Vdash_d \nabla\alpha = \mathsf{e}$ and $d(b)\ \widehat{T}(\Vdash_d)\ \alpha = \mathsf{e}$. It suffices to show that then $d(b)\ \widehat{T}(\mathcal{I}(\Vdash_d^{\geq \mathsf{e}}))\ \alpha = \mathsf{e}$, to conclude that $\widehat{T}(R)(c(a), d(b)) = \mathsf{e}$ using the second diagram. To that end we use the assumption that $\mathscr{V}$ is a Gödel chain, since then $(\widehat{T}(S))^{\geq \mathsf{e}} \leq \widehat{T}(\mathcal{I}(S^{\geq \mathsf{e}}))$ holds for any $\mathscr{V}$-relation S: suppose that $\widehat{T}(S)(\alpha,\beta) \geq \mathsf{e}$. This means that $\bigvee_{t_\alpha, t_\beta} \widehat{H_\Sigma} S(t_\alpha, t_\beta) \geq \mathsf{e}$. Since there is only finitely many terms representing α and β, and sice $\mathscr{V}$ is a chain, this entails that there are terms t_α, t_β with $\widehat{H_\Sigma} S(t_\alpha, t_\beta) \geq \mathsf{e}$, proving that $\widehat{T}(\mathcal{I}(S^{\geq \mathsf{e}}))(\alpha,\beta)$.

We have explicitly used that the relation lifting is 2-functorial, restricts to bases, commutes with graphs and opposites of relations, and the fact that the base is finite and $c(a) \in T\mathsf{Base}(c(a))$ and $d(b) \in T\mathsf{Base}(d(b))$.

Part 2 of Theorem 35 indeed entails expressivity: if a and b are not $\mathscr{V}$-bisimilar, then they are not bisimilar, therefore $R(a,b)$ does not hold and there exists a formula $\varphi \in \mathcal{L}$ such that $(a \Vdash_c \varphi) \neq (b \Vdash_d \varphi)$. □

5 Conclusion and Further Work

We have shown that one can successfully work with many-valued relations, their lifting, and resulting notions of bisimilarity and Moss' coalgebraic language, while in the category of sets. In this sense our results are counterpart to those obtained in [5] in the enriched setting of $\mathscr{V}$-categories.

We can perceive our definition of Moss' $\mathscr{V}$-logic as a definition of a minimal coalgebraic modal logic over the logic of the algebra $\mathscr{V}$, a task closely related to the motivation behind the work of Bou et. al in [6].

There are several obvious directions of further research of the Moss' $\mathscr{V}$-logic, namely to find an axiomatization of the logic, and to prove a completeness result. Although so far the material has been (up to the technicalities) quite similar to the case of the standard two-valued setting, with the axiomatization it will obviously not be so. The main reason we see for that is that we have significantly complicated the propositional part of the logic, which in general is the logic of $\mathscr{V}$, for which the problem of axiomatization could already be hard. The task proved difficult already in a more standard setting of many-valued modal logics of [6]. The first step therefore would be to axiomatize the modal part only, relatively over an assumed axiomatization of the logic of $\mathscr{V}$, for some particular examples of Heyting algebras, namely the finite ones.

We expect it is possible to extend our results to all functors preserving weak pullbacks (dropping the finitarity requirement) using the "cut" approach. Also the respective notion of distributive laws should be studied in our setting.

The nabla modality in the boolean context can be used to produce a disjunctive normal form for standard modal logic, the generalisation we introduced may give rise to certain normal forms in the logic of Fitting [8,9]. This connection is yet to be understood.

References

1. Adámek, J., Gumm, H.P., Trnková, V.: Presentation of set functors: A coalgebraic perspective. Journal of Logic and Computation 20(5), 991–1015 (2010)
2. Adámek, J., Trnková, V.: Automata and Algebras in Categories. Mathematics and its Applications. Kluwer (1990)
3. Barr, M.: Relational algebras. In: Mac Lane, S., Applegate, H., Barr, M., Day, B., Dubuc, E., Phreilambud, Pultr, A., Street, R., Tierney, M., Swierczkowski, S. (eds.) Reports of the Midwest Category Seminar IV. Lecture Notes in Mathematics, vol. 137, pp. 39–55. Springer, Heidelberg (1970)
4. Bílková, M., Kurz, A., Petrişan, D., Velebil, J.: Relation Liftings on Preorders and Posets. In: Corradini, A., Klin, B., Cîrstea, C. (eds.) CALCO 2011. LNCS, vol. 6859, pp. 115–129. Springer, Heidelberg (2011)
5. Bílková, M., Kurz, A., Petrişan, D., Velebil, J.: Relation lifting, with an application to the many-valued cover modality. Log. Methods Comput. Sci. (accepted for publication, 2013)
6. Bou, F., Esteva, F., Godo, L., Rodríguez, R.: On the minimum many-valued modal logic over a finite residuated lattice. Journal of Logic and Computation 21(5), 739–790 (2011)
7. Dostál, M.: Many-valued coalgebraic logic. Master's thesis, Czech Technical University (2013), `http://cyber.felk.cvut.cz/research/theses/papers/322.pdf`
8. Fitting, M.: Many-valued modal logics. In: Fundamenta Informaticae, pp. 365–448. Kluwer Academic Publishers (1992)
9. Fitting, M.: Many-valued modal logics ii. Fundamenta Informaticae 17 (1992)
10. Gumm, H.P.: From T-coalgebras to filter structures and transition systems. In: Fiadeiro, J.L., Harman, N.A., Roggenbach, M., Rutten, J. (eds.) CALCO 2005. LNCS, vol. 3629, pp. 194–212. Springer, Heidelberg (2005)
11. Kupke, C., Kurz, A., Venema, Y.: Completeness for the coalgebraic cover modality. Logical Methods in Computer Science 8(3) (2012)
12. Kurz, A., Leal, R.: Modalities in the stone age: A comparison of coalgebraic logics. Theoretical Computer Science 430, 88–116 (2012); Mathematical Foundations of Programming Semantics (MFPS XXV)
13. Moss, L.: Coalgebraic logic. Annals of Pure and Applied Logic 96 (1999)
14. Trnková, V.: Relational automata in a category and their languages. In: Karpinski, M. (ed.) FCT 1977. LNCS, vol. 56, pp. 340–355. Springer, Heidelberg (1977)

Saturated Semantics for Coalgebraic Logic Programming

Filippo Bonchi and Fabio Zanasi

ENS Lyon, U. de Lyon, CNRS, INRIA, UCBL, France

Abstract. A series of recent papers introduces a coalgebraic semantics for logic programming, where the behavior of a goal is represented by a *parallel* model of computation called coinductive tree. This semantics fails to be compositional, in the sense that the coalgebra formalizing such behavior does not commute with the substitutions that may apply to a goal. We suggest that this is an instance of a more general phenomenon, occurring in the setting of interactive systems (in particular, nominal process calculi), when one tries to model their semantics with coalgebrae on presheaves. In those cases, compositionality can be obtained through *saturation*. We apply the same approach to logic programming: the resulting semantics is compositional and enjoys an elegant formulation in terms of coalgebrae on presheaves and their right Kan extensions.

1 Introduction

Coalgebrae on presheaves have been successfully employed to provide semantics to *nominal* calculi: sophisticated process calculi with complex mechanisms for variable binding, like the π-calculus [11,12]. The idea is to have an index category $\mathbf{C}$ of interfaces (or names), and encode as a presheaf $\mathcal{F}\colon \mathbf{C} \to \mathbf{Set}$ the mapping of any object i of $\mathbf{C}$ to the set of states having i as interface, and any arrow $f\colon i \to j$ to a function switching the interface of states from i to j. The operational semantics of the calculus will arise as a notion of transition between states, that is, as a coalgebra $\alpha\colon \mathcal{F} \to \mathcal{B}(\mathcal{F})$, where $\mathcal{B}\colon \mathbf{Set}^{\mathbf{C}} \to \mathbf{Set}^{\mathbf{C}}$ is a functor on presheaves encoding the kind of behavior that we want to express.

As an arrow in a presheaf category, α has to be a natural transformation, i.e. it should commute with arrows $f\colon i \to j$ in the index category $\mathbf{C}$. Unfortunately, this naturality requirement may fail when the structure of $\mathbf{C}$ is rich enough, as for instance when non-injective substitutions [21,24] or name fusions [20,3] occur. As a concrete example, consider the π-calculus term $t = \bar{a}\langle x\rangle | b(y)$ consisting of a process $\bar{a}\langle x\rangle$ sending a message x on a channel named a, in parallel with $b(y)$ receiving a message on a channel named b. Since the names a and b are different, the two processes cannot synchronize. Conversely the term $t\theta = \bar{a}\langle x\rangle | a(y)$, that is obtained by applying the substitution θ mapping b to a, can synchronize. If θ is an arrow of the index category $\mathbf{C}$, then the operational semantics α is not natural since $\alpha(t\theta) \neq \alpha(t)\overline{\theta}$, where $\overline{\theta}$ denotes the application of θ to the transitions of t. As a direct consequence, also the unique morphism to the

R. Heckel and S. Milius (Eds.): CALCO 2013, LNCS 8089, pp. 80–94, 2013.

terminal coalgebra is not natural: this means that the abstract semantics of π-calculus is not *compositional* - in other words, bisimilarity is not a congruence w.r.t. name substitutions. In order to make bisimilarity a congruence, Sangiorgi introduced in [23] *open bisimilarity*, that is defined by considering the transitions of processes under *all* possible name substitutions θ.

The approach of *saturated semantics* [5] can be seen as a generalization of open bisimilarity, relying on analogous principles: the operational semantics α is "saturated" w.r.t. the arrows of the index category $\mathbf{C}$, resulting in a natural transformation $\alpha^\sharp$ in $\mathbf{Set}^{\mathbf{C}}$. In [3,22], this is achieved by first shifting the definition of α to the category $\mathbf{Set}^{|\mathbf{C}|}$ of presheaves indexed by the discretization $|\mathbf{C}|$ of $\mathbf{C}$. Since $|\mathbf{C}|$ does not have other arrow than the identities, α is trivially a natural transformation in this setting. The source of α is $\mathcal{U}(\mathcal{F}) \in \mathbf{Set}^{|\mathbf{C}|}$, where $\mathcal{U}\colon \mathbf{Set}^{\mathbf{C}} \to \mathbf{Set}^{|\mathbf{C}|}$ is a forgetful functor defined by composition with the inclusion $\iota\colon |\mathbf{C}| \to \mathbf{C}$. The functor $\mathcal{U}$ has a right adjoint $\mathcal{K}\colon \mathbf{Set}^{|\mathbf{C}|} \to \mathbf{Set}^{\mathbf{C}}$ sending a presheaf to its *right Kan extension* along ι. The adjoint pair $\mathcal{U} \dashv \mathcal{K}$ induces an isomorphism $(-)^\sharp_{X,Y}\colon \mathbf{Set}^{|\mathbf{C}|}[\mathcal{U}(X), Y] \to \mathbf{Set}^{\mathbf{C}}[X, \mathcal{K}(Y)]$ mapping α to $\alpha^\sharp$. The latter is a natural transformation in $\mathbf{Set}^{\mathbf{C}}$ and, consequently, the abstract semantics results to be compositional.

In this paper, we show that the saturated approach can be fruitfully instantiated to *coalgebraic logic programming* [16,18,17], which consists of a novel semantics for logic programming and a parallel resolution algorithm based on *coinductive trees*. These are a variant of and-or trees [14] modeling *parallel* implementations of logic programming, where the soundness of the derivations represented by a tree is guaranteed by the restriction to *term-matching* (whose algorithm, differently from unification, is parallelizable [10]).

There are two analogies with the π-calculus: (a) the state space is modeled by a presheaf on the index category $\mathbf{L}_{\mathbf{\Sigma}}^{op}$, that is the (opposite) *Lawvere Theory* associated with some signature Σ; (b) the operational semantics given in [18] fails to be a natural transformation in $\mathbf{Set}^{\mathbf{L}_{\mathbf{\Sigma}}^{op}}$: Example 2 provides a counter-example which is similar to the π-calculus term t discussed above.

The authors of [18] obviate to (b) by relaxing naturality to *lax naturality*: the operational semantics p of a logic program is given as an arrow in the category $Lax(\mathbf{L}_{\mathbf{\Sigma}}^{op}, \mathbf{Poset})$ of locally ordered functors $\mathcal{F}\colon \mathbf{L}_{\mathbf{\Sigma}}^{op} \to \mathbf{Poset}$ and lax natural transformations between them. They show the existence of a cofree comonad that induces a morphism $[\![-]\!]_p$ mapping atoms (i.e., atomic formulae) to coinductive trees. Since $[\![-]\!]_p$ is not natural but lax natural, the semantics provided by coinductive trees is not compositional, in the sense that, for some atoms A and substitution θ,

$$[\![A\theta]\!]_p \neq [\![A]\!]_p \overline{\theta}$$

where $[\![A\theta]\!]_p$ is the coinductive tree associated with $A\theta$ and $[\![A]\!]_p \overline{\theta}$ denotes the result of applying θ to each atom occurring in the tree $[\![A]\!]_p$.

Instead of introducing laxness, we propose to tackle the non-naturality of p with a saturated approach. It turns out that, in the context of logic programming, the saturation map $(-)^\sharp$ has a neat description in terms of substitution

mechanisms: while p performs *term-matching* between the atoms and the heads of clauses of a given logic program, its saturation $p^\sharp$ (given as a morphism in $\mathbf{Set}^{\mathbf{L}_\Sigma^{op}}$) performs *unification*. It is worth to remark here that not only most general unifiers are considered but *all* possible unifiers.

A cofree construction leading to a map $[\![-]\!]_{p^\sharp}$ can be obtained by very standard categorical tools, such as terminal sequences [1]. This is possible because, as $\mathbf{Set}$, both $\mathbf{Set}^{\mathbf{L}_\Sigma^{op}}$ and $\mathbf{Set}^{|\mathbf{L}_\Sigma^{op}|}$ are (co)complete categories, whereas in the lax approach, $Lax(\mathbf{L}_\Sigma^{op}, \mathbf{Poset})$ not being (co)complete, more indirect and more sophisticated categorical constructions are needed [17, Sec. 4]. By naturality of $p^\sharp$, the semantics given by $[\![-]\!]_{p^\sharp}$ turns out to be compositional, as in the desiderata. Analogously to $[\![-]\!]_p$, also $[\![-]\!]_{p^\sharp}$ maps atoms to tree structures, which we call *saturated trees*. They generalize coinductive trees, in the sense that the latter can be seen as a "desaturation" of saturated trees, where all unifiers that are not term-matchers have been discarded. This observation leads to a *translation* from saturated to coinductive trees, based on the counit ϵ of the adjunction $\mathcal{U} \dashv \mathcal{K}$. It follows that our framework encompasses the semantics in [18,17].

Analogously to what is done in [17], we propose a notion of *refutation subtree* of a given saturated tree, intuitively corresponding to an SLD-refutation of an atomic goal in a program. This leads to a result of soundness and completeness of our semantics with respect to SLD-resolution, crucially using both compositionality and the translation into coinductive trees.

Related works. Apart from [16,18,17], there exist other categorical perspectives on (extensions of) logic programming, such as [9,15,2]. Amongst these, the most relevant for us is [5] since it exploits a form of saturation: states representing formulae are both instantiated by substitution and contextualized by other formulae in "and". Beyond logic programming, the idea of exploiting saturation to achieve compositionality is even older than [23]. As far as we know, [8] is the first work where saturation is explored in terms of coalgebrae. It is interesting to note that, in [7], a subset of the same authors also proposed laxness as a solution for the lack of compositionality of Petri nets.

A third approach, alternative to laxness and saturation, may be possible by taking a special kind of "powerobject" functor as done in [20,24] for giving a coalgebraic semantics to fusion and open π-calculus. We have chosen saturated semantics for its generality: it works for any behavioral functor $\mathcal{B}$ and it models a phenomenon that occurs in many different computational models (see e.g. [4]).

Acknowledgements. We thank E. Komendantskaya, T. Hirschowitz, D. Petrisan, J. Power, M. Sammartino, the *Plume* team and the anonymous referees for the helpful comments. Our work is supported by project ANR 12IS02001 PACE.

2 Coalgebraic Logic Programming

In this section we recall the framework of coalgebraic logic programming, as introduced in [16,18,17]. For this purpose, we first fix some terminology and notation, mainly concerning category theory and logic programming.

Given a (small) category $\mathbf{C}$, $|\mathbf{C}|$ denotes the category with the same objects as $\mathbf{C}$ but no other arrow than the identities. With a little abuse of notation, $o \in |\mathbf{C}|$ indicates that o is an object of $\mathbf{C}$ and $\mathbf{C}[o_1, o_2]$ the set of arrows from o_1 to o_2. A $\mathbf{C}$-indexed *presheaf* is any functor $\mathcal{G}\colon \mathbf{C} \to \mathbf{Set}$. We write $\mathbf{Set}^{\mathbf{C}}$ for the category of $\mathbf{C}$-indexed presheaves and natural transformations between them. Given a functor $\mathcal{B}\colon \mathbf{C} \to \mathbf{C}$, a $\mathcal{B}$-*coalgebra* on $o \in |\mathbf{C}|$ is an arrow $p\colon o \to \mathcal{B}(o)$.

We fix a *signature* Σ of function symbols, each equipped with a fixed arity, and a countably infinite set $Var = \{x_1, x_2, x_3, \dots\}$ of variables. We model substitutions and unification of terms over Σ and Var according to the categorical perspective of [13,6]. To this aim, let the (opposite) *Lawvere Theory* of Σ be a category $\mathbf{L}_{\mathbf{\Sigma}}^{op}$ where objects are natural numbers, with $n \in |\mathbf{L}_{\mathbf{\Sigma}}^{op}|$ intuitively representing variables $x_1, x_2, \dots, x_n$ from Var. For any two $n, m \in |\mathbf{L}_{\mathbf{\Sigma}}^{op}|$, the set $\mathbf{L}_{\mathbf{\Sigma}}^{op}[n, m]$ consists of all n-tuples $\langle t_1, \dots, t_n \rangle$ of terms where only variables among $x_1, \dots, x_m$ occur. The identity on $n \in |\mathbf{L}_{\mathbf{\Sigma}}^{op}|$, denoted by id_n, is given by the tuple $\langle x_1, \dots, x_n \rangle$. The composition of $\langle t_1^1, \dots, t_n^1 \rangle\colon n \to m$ and $\langle t_1^2, \dots, t_m^2 \rangle\colon m \to m'$ is the tuple $\langle t_1, \dots, t_n \rangle\colon n \to m'$, where t_i is the term t_i^1 in which every variable x_j has been replaced with t_j^2, for $1 \leq j \leq m$ and $1 \leq i \leq n$.

We call *substitutions* the arrows of $\mathbf{L}_{\mathbf{\Sigma}}^{op}$ and use Greek letters θ, σ and τ to denote them. Given $\theta_1\colon n \to m_1$ and $\theta_2\colon n \to m_2$, a *unifier* of θ_1 and θ_2 is a pair of substitutions $\sigma\colon m_1 \to m$ and $\tau\colon m_2 \to m$, where m is some object of $\mathbf{L}_{\mathbf{\Sigma}}^{op}$, such that $\sigma \circ \theta_1 = \tau \circ \theta_2$. The *most general unifier* of θ_1 and θ_2 is a unifier with a universal property, i.e. a pushout of the diagram $m_1 \xleftarrow{\theta_1} n \xrightarrow{\theta_2} m_2$.

An *alphabet* $\mathcal{A}$ consists of a signature Σ, a set of variables Var and a set of predicate symbols $P, P_1, P_2, \dots$ each assigned an arity. Given P of arity n and Σ-terms $t_1, \dots, t_n$, $P(t_1, \dots, t_n)$ is called an *atom*. We use Latin capital letters $A, B, \dots$ for atoms. Given a substitution $\theta = \langle t_1, \dots, t_n \rangle\colon n \to m$ and an atom A with variables among $x_1, \dots, x_n$, we adopt the standard notation of logic programming in denoting with $A\theta$ the atom obtained by replacing x_i with t_i in A, for $1 \leq i \leq n$. The atom $A\theta$ is called a *substitution instance* of A. The notation $\{A_1, \dots, A_m\}\theta$ is a shorthand for $\{A_1\theta, \dots, A_m\theta\}$. Given atoms A_1 and A_2, we say that A_1 *unifies* with A_2 (equivalently, they are *unifiable*) if they are of the form $A_1 = P(t_1, \dots, t_n)$, $A_2 = P(t'_1, \dots, t'_n)$ and a unifier $\langle \sigma, \tau \rangle$ of $\langle t_1, \dots, t_n \rangle$ and $\langle t'_1, \dots, t'_n \rangle$ exists. Observe that, by definition of unifier, this amounts to saying that $A_1\sigma = A_2\tau$. *Term matching* is a particular case of unification, where σ is the identity substitution. In this case we say that $\langle \sigma, \tau \rangle$ is a *term-matcher* of A_1 and A_2, meaning that $A_1 = A_2\tau$.

A *logic program* $\mathbb{P}$ consists of a finite set of *clauses* C written as $H \leftarrow B_1, \dots, B_k$. The components H and $B_1, \dots, B_k$ are atoms, where H is called the *head* of C and $B_1, \dots, B_k$ form the *body* of C. One can think of $H \leftarrow B_1, \dots, B_k$ as representing the first-order formula $(B_1 \wedge \dots \wedge B_k) \rightarrow H$. We say that $\mathbb{P}$ is *ground* if only ground atoms (i.e. without variables) occur in its clauses. The central algorithm of logic programming is SLD-resolution, checking whether a finite set of atoms (called a *goal*) is *refutable* in $\mathbb{P}$ and giving a substitution called *computed answer* as output. Relevant for our exposition are *and-or trees* [14], which represent executions of SLD-resolution exploiting two forms of parallelism: *and-parallelism*,

corresponding to simultaneous refutation-search of multiple atoms in a goal, and *or-parallelism*, exploring multiple attempts to refute the same goal.

Definition 1. *Given a logic program $\mathbb{P}$ and an atom A, the* (parallel) and-or tree *for A in $\mathbb{P}$ is the possibly infinite tree T satisfying the following properties:*

1. *Each node in T is either an and-node or an or-node.*
2. *Each and-node is labeled with one atom and its children are or-nodes.*
3. *The root of T is an and-node labeled with A.*
4. *Each or-node is labeled with $\bullet$ and its children are and-nodes.*
5. *For every and-node s in T, let A' be its label. For every clause $H \leftarrow B_1, \ldots, B_k$ of $\mathbb{P}$ and most general unifier $\langle \sigma, \tau \rangle$ of A' and H, s has exactly one child t, and viceversa. For each atom B in $\{B_1, \ldots, B_k\}\tau$, t has exactly one child labeled with B, and viceversa.*

As standard for any tree, we have a notion of *depth*: the root is at depth 0 and depth $i+1$ is given by the children of nodes at depth i.

2.1 The Ground Case

We recall the coalgebraic semantics of ground logic programs introduced in [16]. For the sequel we fix an alphabet $\mathcal{A}$, a set At of ground atoms and a ground logic program $\mathbb{P}$. The behavior of $\mathbb{P}$ is represented by a coalgebra $p\colon At \to \mathcal{P}_f\mathcal{P}_f(At)$ on **Set**, where $\mathcal{P}_f$ is the finite powerset functor and p is defined as follows:

$$p\colon\ A \ \mapsto\ \{\{B_1, \ldots, B_k\} \mid H \leftarrow B_1, \ldots, B_k \text{ is a clause of } \mathbb{P} \text{ and } A = H\}.$$

The idea is that p maps an atom $A \in At$ to the set of bodies of clauses of $\mathbb{P}$ whose head H unifies with A, i.e. (in the ground case) $A = H$. Therefore $p(A) \in \mathcal{P}_f\mathcal{P}_f(At)$ can be seen as representing the and-or tree of A in $\mathbb{P}$ up to depth 2, according to Definition 1: each element $\{B_1, \ldots, B_k\}$ of $p(A)$ corresponds to a child of the root, whose children are labeled with $B_1, \ldots, B_k$. The full tree is recovered as an element of $\mathcal{C}(\mathcal{P}_f\mathcal{P}_f)(At)$, where $\mathcal{C}(\mathcal{P}_f\mathcal{P}_f)$ is the *cofree comonad* on $\mathcal{P}_f\mathcal{P}_f$, standardly provided by the following construction [1,25].

Construction 1. *The terminal sequence for the functor $At \times \mathcal{P}_f\mathcal{P}_f(-)\colon \mathbf{Set} \to \mathbf{Set}$ consists of sequences of objects X_α and arrows $\delta_\alpha\colon X_{\alpha+1} \to X_\alpha$, defined by induction on α as follows.*

$$X_\alpha := \begin{cases} At & \alpha = 0 \\ At \times \mathcal{P}_f\mathcal{P}_f(X_\beta) & \alpha = \beta + 1 \end{cases} \qquad \delta_\alpha := \begin{cases} \pi_1 & \alpha = 0 \\ id_{At} \times \mathcal{P}_f\mathcal{P}_f(\delta_\beta) & \alpha = \beta + 1 \end{cases}$$

For α a limit ordinal, X_α is given as a limit of the sequence and a function $\delta_\alpha\colon X_\alpha \to X_\beta$ is given for each $\beta < \alpha$ by the limiting property of X_α.

By [25] it follows that the sequence given above converges to a limit X_γ such that $X_\gamma \cong X_{\gamma+1}$. Since $X_{\gamma+1}$ is defined as $At \times \mathcal{P}_f\mathcal{P}_f(X_\gamma)$, there is a projection function $\pi_2\colon X_{\gamma+1} \to \mathcal{P}_f\mathcal{P}_f(X_\gamma)$ which makes $\pi_2 \circ \delta_\gamma^{-1}\colon X_\gamma \to \mathcal{P}_f\mathcal{P}_f(X_\gamma)$ the cofree $\mathcal{P}_f\mathcal{P}_f$-coalgebra *on At. This induces the* cofree comonad $\mathcal{C}(\mathcal{P}_f\mathcal{P}_f)\colon \mathbf{Set} \to \mathbf{Set}$ *on $\mathcal{P}_f\mathcal{P}_f$ as a functor mapping At to X_γ.*

As the elements of the cofree comonad on $\mathcal{P}_f$ are standardly presented as finitely branching trees [25], those for $\mathcal{P}_f\mathcal{P}_f$ can be seen as finitely branching trees with two sorts of nodes occurring at alternating depth. We now define a $\mathcal{C}(\mathcal{P}_f\mathcal{P}_f)$-coalgebra $[\![-]\!]_p \colon At \to \mathcal{C}(\mathcal{P}_f\mathcal{P}_f)(At)$.

Construction 2. *Given a ground program $\mathbb{P}$, let $p \colon At \to \mathcal{P}_f\mathcal{P}_f(At)$ be the coalgebra associated with $\mathbb{P}$. We define a cone $\{p_\alpha \colon At \to X_\alpha\}_{\alpha<\gamma}$ on the terminal sequence of Construction 1 as follows:*

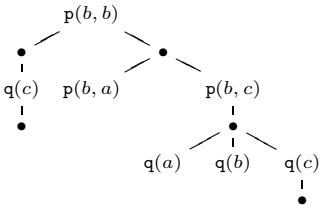
$$p_\alpha := \begin{cases} id_{At} & \alpha = 0 \\ \langle id_{At}, (\mathcal{P}_f\mathcal{P}_f(p_\beta) \circ p)\rangle & \alpha = \beta + 1. \end{cases}$$

For α a limit ordinal, $p_\alpha \colon At \to X_\alpha$ is provided by the limiting property of X_α. Then in particular $X_\gamma = \mathcal{C}(\mathcal{P}_f\mathcal{P}_f)(At)$ yields a function $[\![-]\!]_p \colon At \to \mathcal{C}(\mathcal{P}_f\mathcal{P}_f)(At)$.

Given an atom $A \in At$, the tree $[\![A]\!]_p \in \mathcal{C}(\mathcal{P}_f\mathcal{P}_f)(At)$ is built by iteratively applying the map p, first to A, then to each atom in $p(A)$, and so on. For each natural number m, p_m maps A to its and-or tree up to depth m. As shown in [16], the limit $[\![-]\!]_p$ of all such approximations provides the full and-or tree of A.

Example 1. Consider the ground logic program on the left-hand side, based on an alphabet consisting of a signature $\{a^0, b^0, c^0\}$ and predicates $\mathtt{p}(-,-)$, $\mathtt{q}(-)$. The and-or tree $[\![\mathtt{p}(b,b)]\!]_p \in \mathcal{C}(\mathcal{P}_f\mathcal{P}_f)(At)$ is depicted on the right-hand side.

$\mathtt{p}(b,c) \leftarrow \mathtt{q}(a), \mathtt{q}(b), \mathtt{q}(c)$
$\mathtt{p}(b,b) \leftarrow \mathtt{p}(b,a), \mathtt{p}(b,c)$
$\mathtt{p}(b,b) \leftarrow \mathtt{q}(c)$
$\mathtt{q}(c) \leftarrow$



2.2 The General Case

We recall the extension of the coalgebraic semantics to arbitrary (i.e. possibly non-ground) logic programs presented in [18,17]. In presence of variables, and-or trees are not guaranteed to represent sound derivations, whence *coinductive trees* are introduced as a sound variant of and-or trees, where unification is restricted to term-matching. We refer to [18,17] for more details.

Before formally defining coinductive trees, it is worth recalling that, in [18], the collection of atoms (based on an alphabet $\mathcal{A}$) is modeled as a presheaf $At \colon \mathbf{L}_{\mathbf{\Sigma}}^{op} \to \mathbf{Set}$. The index category is the (opposite) *Lawvere Theory* $\mathbf{L}_{\mathbf{\Sigma}}^{op}$ of Σ, as defined above. For each natural number $n \in |\mathbf{L}_{\mathbf{\Sigma}}^{op}|$, $At(n)$ is defined as the set of atoms with variables among $x_1, \ldots, x_n$. Given an arrow $\theta \in \mathbf{L}_{\mathbf{\Sigma}}^{op}[n,m]$, the function $At(\theta) \colon At(n) \to At(m)$ is defined by substitution, i.e. $At(\theta)(A) := A\theta$. By definition, whenever an atom A belongs to $At(n)$, then it also belongs to $At(n')$, for all $n' \geq n$. However, the occurrences of the same atom in $At(n)$ and $At(n')$ (for $n \neq n'$) are considered distinct: the atoms $A \in At(n)$ and $A \in At(n')$ can be

thought of as two states $x_1, \ldots, x_n \vdash A$ and $x_1, \ldots, x_{n'} \vdash A$ with two different interfaces $x_1, \ldots, x_n$ and $x_1, \ldots, x_{n'}$. For this reason, when referring to an atom A, it is important to always specify the set $At(n)$ to which it belongs.

Definition 2. *Given a logic program $\mathbb{P}$, a natural number n and an atom $A \in At(n)$, the* n-coinductive tree *for A in $\mathbb{P}$ is the possibly infinite tree T satisfying properties 1-4 of Definition 1 and property 5 replaced by the following*[1]*:*

5. *For every and-node s in T, let $A' \in At(n)$ be its label. For every clause $H \leftarrow B_1, \ldots, B_k$ of $\mathbb{P}$ and term-matcher $\langle id_n, \tau \rangle$ of A' and H, with $B_1\tau, \ldots, B_k\tau \in At(n)$, s has exactly one child t, and viceversa. For each atom B in $\{B_1, \ldots, B_k\}\tau$, t has exactly one child labeled with B, and viceversa.*

We recall from [18] the categorical formalization of this class of trees. The first step is to generalize the definition of the coalgebra p associated with a program $\mathbb{P}$. Definition 2 suggests how p should act on an atom $A \in At(n)$, for a fixed n:

$$A \mapsto \{\{B_1, \ldots, B_k\}\tau \mid H \leftarrow B_1, \ldots, B_k \text{ is a clause of } \mathbb{P}, \\ A = H\tau \text{ and } B_1\tau, \ldots, B_k\tau \in At(n)\}. \qquad (1)$$

For each clause $H \leftarrow B_1, \ldots, B_k$, there might be infinitely (but countably) many substitutions τ such that $A = H\tau$ (see e.g. [18]). Thus the object on the right-hand side of (1) will be associated with the functor $\mathcal{P}_c\mathcal{P}_f \colon \mathbf{Set} \to \mathbf{Set}$, where $\mathcal{P}_c$ and $\mathcal{P}_f$ are respectively the countable powerset functor and the finite powerset functor. In order to formalize this as a coalgebra on $At \colon \mathbf{L}_{\mathbf{\Sigma}}^{op} \to \mathbf{Set}$, consider liftings $\widetilde{\mathcal{P}_c} \colon \mathbf{Set}^{\mathbf{L}_{\mathbf{\Sigma}}^{op}} \to \mathbf{Set}^{\mathbf{L}_{\mathbf{\Sigma}}^{op}}$ and $\widetilde{\mathcal{P}_f} \colon \mathbf{Set}^{\mathbf{L}_{\mathbf{\Sigma}}^{op}} \to \mathbf{Set}^{\mathbf{L}_{\mathbf{\Sigma}}^{op}}$, standardly defined on presheaves $\mathcal{F} \colon \mathbf{L}_{\mathbf{\Sigma}}^{op} \to \mathbf{Set}$ by postcomposition respectively with $\mathcal{P}_c$ and $\mathcal{P}_f$. Then one would like to fix (1) as the definition of the n-component of a natural transformation $p \colon At \to \widetilde{\mathcal{P}_c}\widetilde{\mathcal{P}_f}(At)$. The key problem with this formulation is that p would *not* be a natural transformation, as shown by the following example.

Example 2. Consider the signature $\Sigma = \{cons^2, succ^1, zero^0, nil^0\}$ and the predicates $\mathtt{List}(-)$, $\mathtt{Nat}(-)$. The program `NatList`, encoding the definition of lists of natural numbers, will be our running example of a non-ground logic program.

$$\begin{array}{rcl@{\qquad}rcl} \mathtt{List}(cons(x_1, x_2)) & \leftarrow & \mathtt{Nat}(x_1), \mathtt{List}(x_2) & \mathtt{List}(nil) & \leftarrow & \\ \mathtt{Nat}(succ(x_1)) & \leftarrow & \mathtt{Nat}(x_1) & \mathtt{Nat}(zero) & \leftarrow & \end{array}$$

Fix a substitution $\theta = \langle nil \rangle \colon 1 \to 0$ and, for each $n \in |\mathbf{L}_{\mathbf{\Sigma}}^{op}|$, suppose that $p(n) \colon At(n) \to \widetilde{\mathcal{P}_c}\widetilde{\mathcal{P}_f}(At)(n)$ is defined according to (1). Then the square

$$\begin{array}{ccc} At(1) & \xrightarrow{p(1)} & \widetilde{\mathcal{P}_c}\widetilde{\mathcal{P}_f}(At)(1) \\ {\scriptstyle At(\theta)}\downarrow & & \downarrow{\scriptstyle \widetilde{\mathcal{P}_c}\widetilde{\mathcal{P}_f}(At)(\theta)} \\ At(0) & \xrightarrow[p(0)]{} & \widetilde{\mathcal{P}_c}\widetilde{\mathcal{P}_f}(At)(0) \end{array}$$

[1] Our notion of coinductive tree corresponds to the notion of coinductive forest of breadth n as in [17, Def.4.4], the only difference being that we "glue" together all trees of the forest into a single tree.

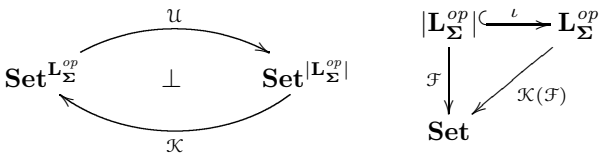
does not commute. A counterexample is provided by the atom $\texttt{List}(x_1) \in At(1)$. Passing through the bottom-left corner of the square, $\texttt{List}(x_1)$ is mapped first to $\texttt{List}(nil) \in At(0)$ and then to $\{\emptyset\} \in \widetilde{\mathcal{P}_c}\widetilde{\mathcal{P}_f}(At)(0)$ - intuitively, this yields a refutation of the goal $\{\texttt{List}(x_1)\}$ with substitution of x_1 with nil. Passing through the top-right corner, $\texttt{List}(x_1)$ is mapped first to $\emptyset \in \widetilde{\mathcal{P}_c}\widetilde{\mathcal{P}_f}(At)(1)$ and then to $\emptyset \in \widetilde{\mathcal{P}_c}\widetilde{\mathcal{P}_f}(At)(0)$, i.e. the computation ends up in a failure.

In [18, Sec.4] the authors overcome this difficulty by relaxing the naturality requirement. The morphism p is defined as a $\breve{\mathcal{P}}_c\breve{\mathcal{P}}_f$-coalgebra in the category $Lax(\mathbf{L}_{\mathbf{\Sigma}}^{op}, \mathbf{Poset})$ of locally ordered functors $\mathcal{F}\colon \mathbf{L}_{\mathbf{\Sigma}}^{op} \to \mathbf{Poset}$ and *lax* natural transformations, with each component $p(n)$ given according to (1) and $\breve{\mathcal{P}}_c\breve{\mathcal{P}}_f$ the extension of $\widetilde{\mathcal{P}_c}\widetilde{\mathcal{P}_f}$ to an endofunctor on $Lax(\mathbf{L}_{\mathbf{\Sigma}}^{op}, \mathbf{Poset})$.

The lax approach fixes the problem, but presents also some drawbacks. Unlike the categories $\mathbf{Set}$ and $\mathbf{Set}^{\mathbf{L}_{\mathbf{\Sigma}}^{op}}$, $Lax(\mathbf{L}_{\mathbf{\Sigma}}^{op}, \mathbf{Poset})$ is neither complete nor cocomplete, meaning that a cofree comonad on $\breve{\mathcal{P}}_c\breve{\mathcal{P}}_f$ cannot be retrieved through the standard Constructions 1 and 2 that were used in the ground case. Moreover, the category of $\breve{\mathcal{P}}_c\breve{\mathcal{P}}_f$-coalgebrae becomes problematic, because coalgebra maps are subject to a commutativity property stricter than the one of lax natural transformations. These two issues force the formalization of non-ground logic program to use quite different (and more sophisticated) categorical tools than the ones employed for the ground case. Finally, as stressed in the Introduction, the laxness of p makes the resulting semantics not compositional.

3 Saturated Semantics

Motivated by the observations of the previous section, we propose a *saturated approach* to the semantics of logic programs. For this purpose, we consider an adjunction between presheaf categories as depicted on the left.



The left adjoint $\mathcal{U}$ is the forgetful functor, given by precomposition with the inclusion functor $\iota\colon |\mathbf{L}_{\mathbf{\Sigma}}^{op}| \hookrightarrow \mathbf{L}_{\mathbf{\Sigma}}^{op}$. As shown in [19, Th.X.1], $\mathcal{U}$ has a right adjoint $\mathcal{K}\colon \mathbf{Set}^{|\mathbf{L}_{\mathbf{\Sigma}}^{op}|} \to \mathbf{Set}^{\mathbf{L}_{\mathbf{\Sigma}}^{op}}$ sending $\mathcal{F}\colon |\mathbf{L}_{\mathbf{\Sigma}}^{op}| \to \mathbf{Set}$ to its *right Kan extension* along ι. This is a presheaf $\mathcal{K}(\mathcal{F})\colon \mathbf{L}_{\mathbf{\Sigma}}^{op} \to \mathbf{Set}$ mapping an object n of $\mathbf{L}_{\mathbf{\Sigma}}^{op}$ to

$$\mathcal{K}(\mathcal{F})(n) := \prod_{\theta \in \mathbf{L}_{\mathbf{\Sigma}}^{op}[n,m]} \mathcal{F}(m)$$

where m is any object of $\mathbf{L}_{\mathbf{\Sigma}}^{op}$. Intuitively, $\mathcal{K}(\mathcal{F})(n)$ is a set of tuples indexed by arrows with source n and such that, at index $\theta\colon n \to m$, there are elements of

$\mathcal{F}(m)$. We use $\dot{x}\ \dot{y}, \ldots$ to denote such tuples and we write $\dot{x}(\theta)$ to denote the element at index θ of the tuple $\dot{x}$. Alternatively, when it is important to show how the elements depend from the indexes, we use $\langle x \rangle_{\theta : n \to m}$ (or simply $\langle x \rangle_\theta$) to denote the tuple having at index θ the element x. With this notation, we can express the behavior of $\mathcal{K}(\mathcal{F})\colon \mathbf{L}_\Sigma^{op} \to \mathbf{Set}$ on an arrow $\theta\colon n \to m$ as

$$\mathcal{K}(\mathcal{F})(\theta)\colon \dot{x} \mapsto \langle \dot{x}(\sigma \circ \theta) \rangle_{\sigma : m \to m'}. \tag{2}$$

The tuple $\langle \dot{x}(\sigma \circ \theta) \rangle_\sigma$ in $\mathcal{K}(\mathcal{F})(m)$ can be intuitively read as follows: for each $\sigma \in \mathbf{L}_\Sigma^{op}[m, m']$, we let the element indexed by σ be the one which was indexed by $\sigma \circ \theta \in \mathbf{L}_\Sigma^{op}[n, m']$ in the input tuple $\dot{x}$.

All this concerns the behavior of $\mathcal{K}$ on the objects of $\mathbf{Set}^{|\mathbf{L}_\Sigma^{op}|}$. For an arrow $f\colon \mathcal{F} \to \mathcal{G}$ in $\mathbf{Set}^{|\mathbf{L}_\Sigma^{op}|}$, the natural transformation $\mathcal{K}(f)$ is defined as an indexwise application of f on tuples from $\mathcal{K}(\mathcal{F})$. For all $n \in |\mathbf{L}_\Sigma^{op}|$, $\dot{x} \in \mathcal{K}(\mathcal{F})(n)$,

$$\mathcal{K}(f)(n)\colon \dot{x} \mapsto \langle f(m)(\dot{x}(\theta)) \rangle_{\theta : n \to m}.$$

For any presheaf $\mathcal{F}\colon \mathbf{L}_\Sigma^{op} \to \mathbf{Set}$, the unit η of the adjunction is instantiated to a morphism $\eta_{\mathcal{F}}\colon \mathcal{F} \to \mathcal{KU}(\mathcal{F})$ given as follows: for all $n \in |\mathbf{L}_\Sigma^{op}|$, $X \in \mathcal{F}(n)$,

$$\eta_{\mathcal{F}}(n)\colon X \mapsto \langle \mathcal{F}(\theta)(X) \rangle_{\theta : n \to m}.$$

When taking $\mathcal{F}$ to be At, $\eta_{At}\colon At \to \mathcal{KU}(At)$ maps an atom to its *saturation*: for each $A \in At(n)$, the tuple $\eta_{At}(n)(A)$ consists of all substitution instances $At(\theta)(A) = A\theta$ of A, each indexed by the corresponding $\theta \in \mathbf{L}_\Sigma^{op}[n, m]$.

As shown in Example 2, given a program $\mathbb{P}$, the family of functions p defined by (1) fails to be a morphism in $\mathbf{Set}^{\mathbf{L}_\Sigma^{op}}$. However, it forms a morphism in $\mathbf{Set}^{|\mathbf{L}_\Sigma^{op}|}$

$$p\colon \mathcal{U}At \to \widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}(\mathcal{U}At)$$

where $\widehat{\mathcal{P}_c}$ and $\widehat{\mathcal{P}_f}$ denote the liftings of $\mathcal{P}_c$ and $\mathcal{P}_f$ to $\mathbf{Set}^{|\mathbf{L}_\Sigma^{op}|}$. The naturality requirement is trivially satisfied in $\mathbf{Set}^{|\mathbf{L}_\Sigma^{op}|}$, since $|\mathbf{L}_\Sigma^{op}|$ is discrete. The adjunction induces a morphism $p^\sharp\colon At \to \mathcal{K}\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\mathcal{U}(At)$ in $\mathbf{Set}^{\mathbf{L}_\Sigma^{op}}$, defined as

$$At \xrightarrow{\eta_{At}} \mathcal{KU}(At) \xrightarrow{\mathcal{K}(p)} \mathcal{K}\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\mathcal{U}(At). \tag{3}$$

In the sequel, we write $\mathcal{S}$ for $\mathcal{K}\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\mathcal{U}$. The idea is to let $\mathcal{S}$ play the same role as $\mathcal{P}_f\mathcal{P}_f$ in the ground case, with the coalgebra $p^\sharp\colon At \to \mathcal{S}(At)$ encoding the program $\mathbb{P}$. An atom $A \in At(n)$ is mapped to $\langle p(m)(A\sigma) \rangle_{\sigma : n \to m}$, that is:

$$p^\sharp(n)\colon A \mapsto \langle \{\{B_1, \ldots, B_k\}\tau \mid H \leftarrow B_1, \ldots, B_k \text{ is a clause of } \mathbb{P}, \\ A\sigma = H\tau \text{ and } B_1\tau, \ldots, B_k\tau \in At(m)\} \rangle_{\sigma : n \to m}. \tag{4}$$

Intuitively, $p^\sharp(n)$ retrieves all unifiers $\langle \sigma, \tau \rangle$ of A and heads of $\mathbb{P}$: first, $A\sigma \in At(m)$ arises as a component of the saturation of A, according to $\eta_{At}(n)$; then, the substitution τ is given by term-matching on $A\sigma$, according to $K(p)(m)$.

By naturality of $p^\sharp$, we achieve the property of "commuting with substitutions" that was precluded by the term-matching approach, as shown by the following rephrasing of Example 2.

Example 3. Consider the same square of Example 2, with $p^\sharp$ in place of p and $\mathcal{S}$ in place of $\widetilde{\mathcal{P}_c}\widetilde{\mathcal{P}_f}$. The atom $\mathtt{List}(x_1) \in At(1)$ together with the substitution $\theta = \langle nil \rangle \colon 1 \to 0$ does not constitute a counterexample to commutativity anymore. Indeed $p^\sharp(1)$ maps $\mathtt{List}(x_1)$ to the tuple $\langle p(n)(\mathtt{List}(x_1)\sigma)\rangle_{\sigma \colon 1 \to n}$, which is then mapped by $\mathcal{S}(At)(\theta)$ to $\langle p(n)(\mathtt{List}(x_1)\sigma' \circ \theta)\rangle_{\sigma' \colon 0 \to n}$ according to (2). Observe that the latter is just the tuple $\langle p(n)(\mathtt{List}(nil)\sigma')\rangle_{\sigma' \colon 0 \to n}$ obtained by applying first $At(\theta)$ and then $p^\sharp(0)$ to the atom $\mathtt{List}(x_1)$.

Another benefit of saturated semantics is that $p^\sharp \colon At \to \mathcal{S}(At)$ lives in a (co)complete category which behaves (pointwise) as **Set**. This allows us to follow the same steps as in the ground case, constructing a coalgebra for the cofree comonad $\mathcal{C}(\mathcal{S})$ as a straightforward generalization of Constructions 1 and 2.

Construction 3. *The terminal sequence for the functor $At \times \mathcal{S}(-) \colon \mathbf{Set}^{\mathbf{L}_\Sigma^{op}} \to \mathbf{Set}^{\mathbf{L}_\Sigma^{op}}$ consists of a sequence of objects X_α and arrows $\delta_\alpha \colon X_{\alpha+1} \to X_\alpha$, which are defined just as in Construction 1, with $\mathcal{S}$ replacing $\mathcal{P}_f\mathcal{P}_f$. By using [25, Th.7], it can be checked that this sequence converges to a limit X_γ such that $X_\gamma \cong X_{\gamma+1}$ and X_γ is the carrier of the cofree $\mathcal{S}$-coalgebra on At.*

Since $\mathcal{S}$ is accessible, the *cofree comonad* $\mathcal{C}(\mathcal{S})$ exists and maps At to X_γ given as in Construction 3. A $\mathcal{C}(\mathcal{S})$-coalgebra $[\![-]\!]_{p^\sharp} \colon At \to \mathcal{C}(\mathcal{S})(At)$ is given below.

Construction 4. *The terminal sequence for $At \times \mathcal{S}(-)$ induces a cone $\{p^\sharp_\alpha \colon At \to X_\alpha\}_{\alpha<\gamma}$ as in Construction 2 with $p^\sharp$ and $\mathcal{S}$ replacing p and $\mathcal{P}_f\mathcal{P}_f$. This yields a natural transformation $[\![-]\!]_{p^\sharp} \colon At \to X_\gamma$, where $X_\gamma = \mathcal{C}(\mathcal{S})(At)$.*

As in the ground case, the coalgebra $[\![-]\!]_{p^\sharp}$ is constructed as an iterative application of $p^\sharp$: we call *saturated tree* the associated tree structure.

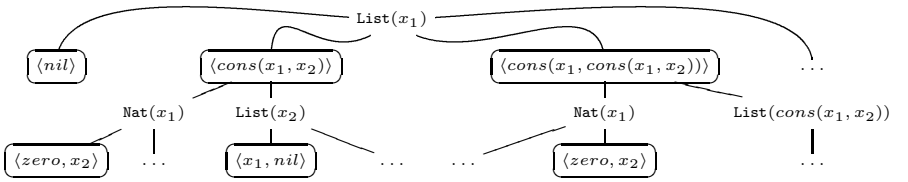
Definition 3. *Given a logic program $\mathbb{P}$, a natural number n and an atom $A \in At(n)$, the* saturated tree *for A in $\mathbb{P}$ is the possibly infinite tree T satisfying properties 1-3 of Definition 1 and properties 4 and 5 replaced by the following:*

4. *Each or-node is labeled with a substitution σ and its children are and-nodes.*
5. *For every and-node s in T, let $A' \in At(n')$ be its label. For every clause $H \leftarrow B_1, \ldots, B_k$ of $\mathbb{P}$ and unifier $\langle \sigma, \tau \rangle$ of A' and H, with $\sigma \colon n' \to m'$ and $B_1\tau, \ldots, B_k\tau \in At(m')$, s has exactly one child t labeled with σ, and viceversa. For each atom B in $\{B_1, \ldots, B_k\}\tau$, t has exactly one child labeled with B, and viceversa.*

We have now seen three kinds of tree, exhibiting different substitution mechanisms. In saturated trees one considers all the unifiers, whereas in and-or trees and coinductive trees one restricts to most general unifiers and term-matchers respectively. Moreover, in a coinductive tree each and-node is labeled with an atom in $At(n)$ for a fixed n, while in a saturated tree n can dynamically change.

Example 4. Part of the infinite saturated tree of $\mathtt{List}(x_1) \in At(1)$ in `NatList` is depicted below. Note that not all labels of and-nodes belong to $At(1)$, as it would be the case for a coinductive tree: such information is inherited from the label of

the parent or-node, which is now a substitution. For instance, both $\mathtt{Nat}(x_1)$ and $\mathtt{List}(x_2)$ belong to $At(2)$, since their parent is labeled with $\langle cons(x_1, x_2)\rangle \colon 1 \to 2$ (using the convention that the target of a substitution is the largest index appearing among its variables).



We can generalize these observations to the following adequacy theorem.

Theorem 1. *Let $[\![-]\!]_{p^\sharp}$ be defined from a program $\mathbb{P}$ according to Construction 4. Then, for all n and $A \in At(n)$, the saturated tree of A in $\mathbb{P}$ is $[\![A]\!]_{p^\sharp}$.*

In the above theorem and in the rest of the paper, with an abuse of notation we use $[\![A]\!]_{p^\sharp}$ to denote the application of $[\![-]\!]_{p^\sharp}(n)$ to $A \in At(n)$ without mentioning the object $n \in |\mathbf{L}_\Sigma^{op}|$. For an arrow $\theta \in \mathbf{L}_\Sigma^{op}[n, m]$, we write $\overline{\theta}$ for $\mathcal{C}(\mathcal{S})(At)(\theta) \colon \mathcal{C}(\mathcal{S})(At)(n) \to \mathcal{C}(\mathcal{S})(At)(m)$. With this notation, we can state the following theorem that is an immediate consequence of the naturality of $[\![-]\!]_{p^\sharp}$.

Theorem 2 (Compositionality). *For all atoms $A \in At(n)$ and substitutions $\theta \in \mathbf{L}_\Sigma^{op}[n, m]$,*

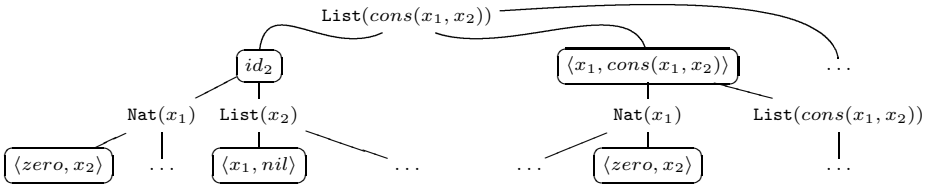
$$[\![A\theta]\!]_{p^\sharp} = [\![A]\!]_{p^\sharp}\overline{\theta}.$$

We conclude this section with a concrete description of the behavior of the operator $\overline{\theta}$, for a given substitution $\theta \in \mathbf{L}_\Sigma^{op}[n, m]$. Let r be the root of a tree $T \in \mathcal{C}(\mathcal{S})(At)(n)$ and r' the root of $T\overline{\theta}$. Then

1. the node r has label A iff r' has label $A\theta$;
2. the node r has a child t with label $\sigma \circ \theta$ and children $t_1, \ldots, t_n$ iff r' has a child t' with label σ and children $t_1 \ldots t_n$.

Note that the children $t_1, \ldots, t_n$ are exactly the same in both trees: $\overline{\theta}$ only modifies the root and the or-nodes at depth 1 of T, while it leaves untouched all the others. This peculiar behavior can be better understood by observing that the definition of $\mathcal{K}(\mathcal{F})(\theta)$, as in (2), is independent of the presheaf $\mathcal{F}$. As a result, $\overline{\theta} = X_\gamma(\theta)$ is independent of all the X_αs built in Construction 3.

Example 5. Recall from Example 4 the saturated tree $[\![\mathtt{List}(x_1)]\!]_{p^\sharp}$. For $\theta = \langle cons(x_1, x_2)\rangle$, the tree $[\![\mathtt{List}(x_1)]\!]_{p^\sharp}\overline{\theta}$ is depicted below.



4 Desaturation

One of the main features of coinductive trees is to represent (sound) and-or parallel derivations of goals. This leads the authors of [17] to a resolution algorithm exploiting the two forms of parallelism. Motivated by these developments, we include coinductive trees in our framework, showing how they can be obtained as a "desaturation" of saturated trees.

For this purpose, the key ingredient is given by the *counit* ϵ of the adjunction $\mathcal{U} \dashv \mathcal{K}$. Given a presheaf $\mathcal{F}\colon |\mathbf{L}_{\mathbf{\Sigma}}^{op}| \to \mathbf{Set}$, the morphism $\epsilon_{\mathcal{F}}\colon \mathcal{UK}(\mathcal{F}) \to \mathcal{F}$ is defined as follows: for all $n \in |\mathbf{L}_{\mathbf{\Sigma}}^{op}|$ and $\dot{x} \in \mathcal{UK}(\mathcal{F})(n)$,

$$\epsilon_{\mathcal{F}}(n)\colon \dot{x} \mapsto \dot{x}(id_n) \tag{5}$$

where $\dot{x}(id_n)$ is the element of the input tuple $\dot{x}$ which is indexed by the identity substitution $id_n \in \mathbf{L}_{\mathbf{\Sigma}}^{op}[n, n]$. In the logic programming perspective, the intuition is that, while the unit of the adjunction provides the saturation of an atom, the counit reverses the process. It takes the saturation of an atom and gives back the substitution instance given by the identity, that is, the atom itself.

The next construction defines a morphism $\overline{d}\colon \mathcal{U}\big(\mathcal{C}(\mathcal{S})(At)\big) \to \mathcal{C}(\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f})(\mathcal{U}At)$ where $\mathcal{C}(\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f})\colon \mathbf{Set}^{|\mathbf{L}_{\mathbf{\Sigma}}^{op}|} \to \mathbf{Set}^{|\mathbf{L}_{\mathbf{\Sigma}}^{op}|}$ is the cofree comonad on $\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}$, obtained through a terminal sequence analogously to Construction 3. The idea is that $\overline{d}$ acts on saturated trees as the depthwise application of $\epsilon_{\mathcal{U}At}$.

Construction 5. *For α an ordinal, let us note by Y_α the objects occurring in the construction of $\mathcal{C}(\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f})(\mathcal{U}At)$ and with X_α the ones in the construction of $\mathcal{C}(\mathcal{S})(At)$, converging to $X_\gamma = \mathcal{C}(\mathcal{S})(At)$. We define a sequence $\{d_\alpha\colon \mathcal{U}(X_\alpha) \to Y_\alpha\}_{\alpha<\gamma}$ in $\mathbf{Set}^{|\mathbf{L}_{\mathbf{\Sigma}}^{op}|}$ as follows:*

$$d_\alpha := \begin{cases} id_{\mathcal{U}At} & \alpha = 0 \\ id_{\mathcal{U}At} \times \big(\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}(d_\beta) \circ \epsilon_{\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\mathcal{U}(X_\beta)}\big) & \alpha = \beta + 1. \end{cases}$$

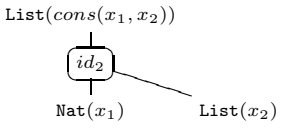
For $\alpha < \gamma$ a limit ordinal, $d_\alpha\colon \mathcal{U}(X_\alpha) \to Y_\alpha$ is provided by the limiting property of Y_α. This sequence induces a morphism $\overline{d}\colon \mathcal{U}\big(\mathcal{C}(\mathcal{S})(At)\big) \to \mathcal{C}(\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f})(\mathcal{U}At)$.

The next theorem shows that $\overline{d}$ is a translation from saturated to coinductive trees: given an atom $A \in At(n)$, it maps $[\![A]\!]_{p^\sharp}$ to the n-coinductive tree of A. The key intuition is that n-coinductive trees can be seen as saturated trees where the labeling of or-nodes has been restricted to the identity substitution id_n, represented as $\bullet$ (see Definition 2). The operation of pruning all or-nodes (and their descendants) in $[\![A]\!]_{p^\sharp}$ which are not labeled with id_n is precisely what is provided by Construction 5, in virtue of the definition of the counit ϵ given in (5).

Theorem 3 (Desaturation). *Let $[\![-]\!]_{p^\sharp}\colon At \to \mathcal{C}(\mathcal{S})(At)$ be defined for a logic program $\mathbb{P}$ according to Construction 4 and $\overline{d}\colon \mathcal{U}\big(\mathcal{C}(\mathcal{S})(At)\big) \to \mathcal{C}(\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f})(\mathcal{U}At)$ be defined according to Construction 5. Then for all $n \in |\mathbf{L}_{\mathbf{\Sigma}}^{op}|$ and $A \in \mathcal{U}At(n)$, the n-coinductive tree of A in $\mathbb{P}$ is $\big(\overline{d} \circ \mathcal{U}([\![-]\!]_{p^\sharp})\big)(n)(A)$.*

Theorem 3 also provides an alternative formalization for the coinductive tree semantics [18], given by composition of the saturated semantics with desaturation. In fact it represents a different approach to the non-compositionality problem: instead of relaxing naturality to lax naturality, we simply forget about all the arrows of the index category $\mathbf{L}_{\mathbf{\Sigma}}^{op}$, shifting the framework from $\mathbf{Set}^{\mathbf{L}_{\mathbf{\Sigma}}^{op}}$ to $\mathbf{Set}^{|\mathbf{L}_{\mathbf{\Sigma}}^{op}|}$. The substitutions on trees (that are essential, for instance, for the resolution algorithm given in [17]) exist at the saturated level, i.e. in $\mathcal{C}(\mathcal{S})(At)$, and they are given precisely as the operator $\overline{\theta}$ described at the end of Section 3.

Example 6. The coinductive tree for $\mathtt{List}(cons(x_1, x_2))$ in `NatList` is depicted on the right. It is constructed by desaturating the tree $[\![\mathtt{List}(cons(x_1, x_2))]\!]_{p^\sharp}$ in Example 5, i.e., by pruning all the or-nodes (and their descendants) that are not labeled with id_2.



5 Soundness and Completeness

The notion of coinductive tree leads to a semantics that is sound and complete with respect to SLD-resolution [17, Th.4.8]. To this aim, a key role is played by *derivation subtrees* of a given coinductive tree.

Definition 4. *Let T be the n-coinductive tree for an atom A in a program $\mathbb{P}$. A subtree T' of T is a* derivation subtree *if it satisfies the following conditions:*

1. *the root of T' is the root of T;*
2. *if an and-node of T belongs to T', then just one of its children belongs to T';*
3. *if an or-node of T belongs to T', then all its children belong to T'.*

A refutation subtree *(called success subtree in [17]) is a finite derivation subtree with only or-nodes as leaves.*

In analogy with coinductive trees, we want to define a notion of subtree for saturated semantics. This requires care: saturated trees are associated with unification, which is more liberal than term-matching. In particular, similarly to and-or trees, they may represent unsound derivation strategies. However, in saturated trees *all* unifiers, and not just the most general ones, are taken into account. This gives enough flexibility to shape a sound notion of subtree, based on an implicit synchronization of the substitutions used in different branches.

Definition 5. *Let T be the saturated tree for an atom A in a program $\mathbb{P}$. A subtree T' of T is called a* synched derivation subtree *if it satisfies properties 1-3 of Definition 4 and the following condition:*

4. *all or-nodes of T' at the same depth are labeled with the same substitution.*

A synched refutation subtree *is a finite synched derivation subtree with only or-nodes as leaves. Its* answer *is the substitution $\theta_{2k+1} \circ \ldots \theta_3 \circ \theta_1$, where θ_i is the (unique) substitution labeling the or-nodes of depth i and $2k+1$ is its maximal depth.*

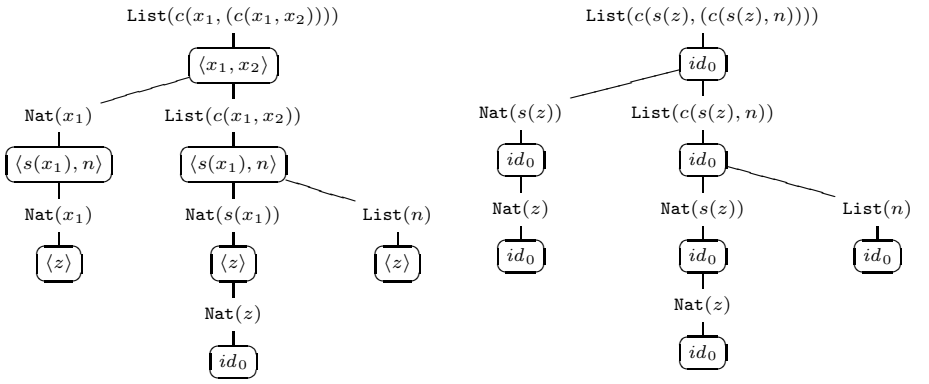


Fig. 1. Successful synched derivation subtrees for $\texttt{List}(cons(x_1, (cons(x_1, x_2))))$ (left) and $\texttt{List}(cons(succ(zero), (cons(succ(zero), nil))))$ (right) in `NatList`. The symbols *cons*, *nil*, *succ* and *zero* are abbreviated to c, n, s and z respectively.

The prefix "synched" emphasizes the restriction to and-parallelism which is encoded in Definition 5. Intuitively, we force all subgoals at the same depth to proceed with *the same* substitution. For instance, this rules out the unsound derivation of [17, Ex.5.2].

Note that derivation subtrees can be seen as special instances of synched derivation subtrees where all the substitutions are forced to be identities.

Theorem 4 (Soundness and Completeness). *Let $\mathbb{P}$ be a logic program and $A \in At(n)$ an atom. The following are equivalent.*

1. *The saturated tree for A in $\mathbb{P}$ has a synched refutation subtree with answer θ.*
2. *There is some natural number m such that the m-coinductive tree for $A\theta$ in $\mathbb{P}$ has a refutation subtree.*
3. *There is an SLD-refutation for $\{A\}$ in $\mathbb{P}$ with computed answer τ such that there exists a substitution σ with $\sigma \circ \tau = \theta$.*

The statement $(2 \Leftrightarrow 3)$ is a rephrasing of [17, Th.4.8], while $(1 \Leftrightarrow 2)$ follows by compositionality and desaturation (Theorems 2 and 3).

Figure 1 provides an example of the argument for direction $(1 \Rightarrow 2)$. Note that the root of the rightmost tree is labeled with an atom of the form $A\theta$, where θ and A are respectively the answer and the label of the root of the leftmost tree. The key observation is that the rightmost tree is a refutation subtree of the 0-coinductive tree for $A\theta$ and can be obtained from the leftmost tree by a procedure involving the operator $\overline{\theta}$ discussed at the end of Section 3.

References

1. Adámek, J., Koubek, V.: On the greatest fixed point of a set functor. Theor. Comput. Sci. 150, 57–75 (1995)
2. Amato, G., Lipton, J., McGrail, R.: On the algebraic structure of declarative programming languages. Theor. Comput. Sci. 410(46), 4626–4671 (2009)

3. Bonchi, F., Buscemi, M.G., Ciancia, V., Gadducci, F.: A presheaf environment for the explicit fusion calculus. J. Autom. Reason. 49(2), 161–183 (2012)
4. Bonchi, F., Montanari, U.: Coalgebraic symbolic semantics. In: Kurz, A., Lenisa, M., Tarlecki, A. (eds.) CALCO 2009. LNCS, vol. 5728, pp. 173–190. Springer, Heidelberg (2009)
5. Bonchi, F., Montanari, U.: Reactive systems (semi-)saturated semantics and coalgebras on presheaves. Theor. Comput. Sci. 410(41), 4044–4066 (2009)
6. Bruni, R., Montanari, U., Rossi, F.: An interactive semantics of logic programming. Theory and Practice of Logic Programming 1(6), 647–690 (2001)
7. Corradini, A., Große-Rhode, M., Heckel, R.: A coalgebraic presentation of structured transition systems. Theor. Comput. Sci. 260(1-2), 27–55 (2001)
8. Corradini, A., Heckel, R., Montanari, U.: From SOS specifications to structured coalgebras: How to make bisimulation a congruence. ENTCS 19, 118–141 (1999)
9. Corradini, A., Montanari, U.: An algebraic semantics for structured transition systems and its application to logic programs. Theor. Comput. Sci. 103(1), 51–106 (1992)
10. Dwork, C., Kanellakis, P.C., Mitchell, J.C.: On the sequential nature of unification. The Journal of Logic Programming 1(1), 35–50 (1984)
11. Fiore, M.P., Moggi, E., Sangiorgi, D.: A fully abstract model for the π-calculus. Inf. Comput. 179(1), 76–117 (2002)
12. Fiore, M.P., Staton, S.: A congruence rule format for name-passing process calculi from mathematical structural operational semantics. In: LICS, pp. 49–58. IEEE (2006)
13. Goguen, J.A.: What is unification? - a categorical view of substitution, equation and solution. In: Resolution of Equations in Algebraic Structures, Volume 1: Algebraic Techniques, pp. 217–261. Academic (1989)
14. Gupta, G., Costa, V.S.: Optimal implementation of and-or parallel prolog. Future Generation Computer Systems 10(1), 71–92 (1994)
15. Kinoshita, Y., Power, A.J.: A fibrational semantics for logic programs. In: Herre, H., Dyckhoff, R., Schroeder-Heister, P. (eds.) ELP 1996. LNCS, vol. 1050, Springer, Heidelberg (1996)
16. Komendantskaya, E., McCusker, G., Power, J.: Coalgebraic semantics for parallel derivation strategies in logic programming. In: Johnson, M., Pavlovic, D. (eds.) AMAST 2010. LNCS, vol. 6486, pp. 111–127. Springer, Heidelberg (2011)
17. Komendantskaya, E., Power, J.: Coalgebraic derivations in logic programming. In: CSL. LIPIcs, vol. 12, pp. 352–366. Schloss Dagstuhl (2011)
18. Komendantskaya, E., Power, J.: Coalgebraic semantics for derivations in logic programming. In: Corradini, A., Klin, B., Cîrstea, C. (eds.) CALCO 2011. LNCS, vol. 6859, pp. 268–282. Springer, Heidelberg (2011)
19. Mac Lane, S.: Categories for the Working Mathematician, 2nd edn. Graduate Texts in Mathematics (September 1998)
20. Miculan, M.: A categorical model of the fusion calculus. ENTCS 218, 275–293 (2008)
21. Miculan, M., Yemane, K.: A unifying model of variables and names. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 170–186. Springer, Heidelberg (2005)
22. Montanari, U., Sammartino, M.: A network-conscious pi-calculus and its coalgebraic semantics. Submitted to TCS (Festschrift for Glynn Winskel)
23. Sangiorgi, D.: A theory of bisimulation for the pi-calculus. Acta Inf. 33(1), 69–97 (1996)
24. Staton, S.: Relating coalgebraic notions of bisimulation. In: Kurz, A., Lenisa, M., Tarlecki, A. (eds.) CALCO 2009. LNCS, vol. 5728, pp. 191–205. Springer, Heidelberg (2009)
25. Worrell, J.: Terminal sequences for accessible endofunctors. ENTCS 19, 24–38 (1999)

Presenting Distributive Laws

Marcello M. Bonsangue[1,3], Helle Hvid Hansen[2,3,⋆], Alexander Kurz[4], and Jurriaan Rot[1,3,⋆⋆]

[1] LIACS – Leiden University, Netherlands
[2] ICIS/IS – Radboud University Nijmegen, Netherlands
[3] Formal Methods – Centrum Wiskunde & Informatica, Amsterdam, Netherlands
[4] Dep. of Computer Science – University of Leicester, United Kingdom

Abstract. Distributive laws of a monad $\mathcal{T}$ over a functor F are categorical tools for specifying algebra-coalgebra interaction. They proved to be important for solving systems of corecursive equations, for the specification of well-behaved structural operational semantics and, more recently, also for enhancements of the bisimulation proof method. If $\mathcal{T}$ is a free monad, then such distributive laws correspond to simple natural transformations. However, when $\mathcal{T}$ is not free it can be rather difficult to prove the defining axioms of a distributive law. In this paper we describe how to obtain a distributive law for a monad with an equational presentation from a distributive law for the underlying free monad. We apply this result to show the equivalence between two different representations of context-free languages.

1 Introduction

The combination of algebraic structure and observable behaviour is fundamental in computer science. Examples include the operational models of structural operational semantics [1], denotational models of programming languages [21], finite stream circuits [12], linear and context-free systems of behavioural differential equations [16,22], and many types of automata such as nondeterministic and weighted automata [18].

In the categorical treatment of these examples, the algebraic structure is encoded by a monad $\mathcal{T} = \langle T, \eta, \mu \rangle$, and the system behaviour by coalgebras for a functor F. Often it is desirable that the algebraic and coalgebraic structure is compatible in some way. A general approach to specifying such algebra-coalgebra interaction is via a distributive law. There are several advantages of this structured approach. A distributive law λ of the monad $\mathcal{T}$ over F induces a $\mathcal{T}$-algebra on the final F-coalgebra of behaviours, yields solutions to corecursive equations $\phi\colon X \to FTX$ [2], and ensures that bisimulation is a congruence. Moreover

⋆ The research of this author has been funded by the Netherlands Organisation for Scientific Research (NWO), Veni project number 639.021.231.
⋆⋆ The research of this author has been funded by the Netherlands Organisation for Scientific Research (NWO), CoRE project, dossier number: 612.063.920.

R. Heckel and S. Milius (Eds.): CALCO 2013, LNCS 8089, pp. 95–109, 2013.

it yields the soundness of techniques such as bisimulation-up-to-context [2] and extensions thereof [14,15].

Describing a distributive law explicitly and proving that it is one can, however, be rather complicated. Therefore, general methods for constructing distributive laws from simpler ingredients are very useful. An important example of this is given by abstract GSOS [19,2,10] where distributive laws of a free monad $\mathcal{T}$ over a (copointed) functor F are shown to correspond to plain natural transformations, called *abstract GSOS-rules* as they can be seen as specification formats. In [3] it was shown how an abstract GSOS-rule for a free monad $\mathcal{T}$ and functor F can be lifted to one for the functor $F(-)^A$ which describes F-systems with input in A. Another method which works for all monads $\mathcal{T}$, but only for certain polynomial behaviour functors F, produces a distributive law inducing a "pointwise lifting" of $\mathcal{T}$-algebra structure to F-behaviours, cf. [4,6,18].

But many examples do not fit into the abovementioned settings. An important motivating example for this paper is that of context-free grammars, where sequential composition is not a pointwise operation and whose formal semantics satisfies the axioms of idempotent semirings, i.e., the algebraic structure is not free. More generally, one may be interested in a monad arising from a free one by adding equations which one knows to hold in the final coalgebra, without having a particular concrete monad in mind.

The main contribution of this paper is to give a general approach for constructing a distributive law λ' for a monad $\mathcal{T}'$ with an equational presentation, from a distributive law λ for the underlying free monad $\mathcal{T}$. We have no constraints on the behaviour functor F. This λ' is obtained as a certain quotient of λ by the equations E of $\mathcal{T}'$, hence we say that λ' *is presented by a* λ *for the free monad and the equations* E. We show that such quotients exist precisely when the distributive law *preserves the equations* E, which roughly means that congruences generated by the equations are bisimulations. We also discuss how these quotients of distributive laws give rise to quotients of bialgebras, thereby giving a concrete operational interpretation, and a correspondence between solutions to corecursive equations with and without equations. As an illustration and application of our theory, we will show the existence of a distributive law of the monad for idempotent semirings over the deterministic automata functor. This result yields the equivalence betweeen the Greibach normal form representation of context-free languages and the coalgebraic representation via context-free expressions given in[22].

Outline. In Section 2 we recall the notions of monads and algebras, and give a concrete description of monad quotients. In Section 3 we recall distributive laws and their application to solving systems of equations. Then in Section 4 we prove our main results on quotients of distributive laws. In Section 5 we show that such quotients give rise to quotients of bialgebras. Finally in Section 6 we discuss related work, and provide some directions for future work. All proofs have been placed in the Appendix.

2 Monads, Algebras and Equations

We start by recalling some basic definitions on monads, algebras, term equations and congruences. We will then proceed to give a concrete description of the quotient monad arising from a free monad and a set of equations. We consider only monads on Set.

A *monad* is a triple $\mathcal{T} = \langle T, \eta, \mu \rangle$ where T is a Set-endofunctor, and $\eta\colon \mathrm{Id} \Rightarrow T$ and $\mu\colon TT \Rightarrow T$ are natural transformations such that $\mu \circ T\eta = 1 = \mu \circ \eta_T$ and $\mu \circ \mu_T = \mu \circ T\mu$. A *$\mathcal{T}$-algebra* is a pair $\langle A, \alpha \rangle$ where A is a set and $\alpha\colon TA \to A$ is a function such that $\alpha \circ \eta_A = 1$ and $\alpha \circ \mu_A = \alpha \circ T\alpha$. A *($\mathcal{T}$-algebra) homomorphism* from $\langle A, \alpha \rangle$ to $\langle B, \beta \rangle$ is a function $f\colon A \to B$ such that $f \circ \alpha = \beta \circ Tf$. The *free $\mathcal{T}$-algebra* over a set X is $\langle TX, \mu_X \rangle$. Given any $\mathcal{T}$-algebra $\langle A, \alpha \rangle$ and any function $f\colon X \to A$, there is a unique algebra homomorphism $f^\sharp\colon TX \to A$ such that $f^\sharp(x) = f(x)$ for all $x \in X$, given by $\alpha \circ Tf$.

Let $\langle T, \eta, \mu \rangle$ and $\langle K, \theta, \nu \rangle$ be monads. A *monad map* is a natural transformation $\sigma\colon T \Rightarrow K$ such that the following diagram commutes:

$$\begin{array}{ccccc} \mathrm{Id} & \xrightarrow{\eta} & T & \xleftarrow{\mu} & TT \\ & \searrow^{\theta} & \downarrow{\scriptstyle \sigma} & & \downarrow{\scriptstyle \sigma\sigma} \\ & & K & \xleftarrow{\nu} & KK \end{array} \tag{1}$$

where $\sigma\sigma = K\sigma \circ \sigma_T = \sigma_K \circ T\sigma$.

We fix a monad $\langle T, \eta, \mu \rangle$, a set of variables V, and a set of *T-equations (over V)* $E \subseteq TV \times TV$. Let $\mathbb{A} = \langle A, \alpha \rangle$ be a $\mathcal{T}$-algebra. We denote by $E_\mathbb{A}$ the relation on A induced by E when quantifying over all valuations $v\colon V \to A$:

$$E_\mathbb{A} = \bigcup_{v\colon V \to A} \{\langle v^\sharp(s), v^\sharp(t) \rangle \mid \langle s, t \rangle \in E\} \tag{2}$$

By $q_\mathbb{A}$ we denote the coequalizer

$$TE_\mathbb{A} \underset{\pi_2^\sharp}{\overset{\pi_1^\sharp}{\rightrightarrows}} A \xrightarrow{q_\mathbb{A}} A/{\equiv_\mathbb{A}}$$

and we write $\equiv_\mathbb{A}$ for the kernel of $q_\mathbb{A}$, i.e.,

$$s \equiv_\mathbb{A} t \qquad \text{iff} \qquad q_\mathbb{A}(s) = q_\mathbb{A}(t)$$

which is the *least congruence* on $\mathbb{A}$ containing $E_\mathbb{A}$. Indeed $q_\mathbb{A}$ is the quotient map of $\equiv_\mathbb{A}$. For a free algebra $\mathbb{A} = \langle TX, \mu_X \rangle$ we write E_X, $\equiv_X$ and q_X for the corresponding equations, congruence and quotient map, respectively. Moreover we let $T'X = TX/{\equiv_X}$. Note that if T is finitary, the quotient is a $\mathcal{T}$-algebra and q is a $\mathcal{T}$-algebra homomorphism.

We will use that all epis in Set are split, so every quotient map q_X has a section $r_X\colon T'X \rightarrowtail TX$ (which picks representatives) such that $q_X \circ r_X = 1_{TX/\equiv}$. The following basic result will be useful.

Lemma 1. *Let $f\colon A \to B$ be an algebra homomorphism from $\mathbb{A} = \langle A, \alpha\rangle$ to $\mathbb{B} = \langle B, \beta\rangle$. Then for all $x, y \in A$: $x \equiv_{\mathbb{A}} y$ implies $f(x) \equiv_{\mathbb{B}} f(y)$.*

Let $f\colon X \to Y$ be any map. Then Tf is an algebra homomorphism from $\langle TX, \mu_X\rangle$ to $\langle TY, \mu_Y\rangle$, so Lemma 1 implies that for any x, y such that $x \equiv_X y$ we have $q_Y(Tf(x)) = q_Y(Tf(y))$. Consequently by the universal property of the coequalizer q_X there is a unique map $T'X \to T'Y$ in the following square:

$$\begin{array}{ccc} TX & \xrightarrow{q_X} & T'X \\ {\scriptstyle Tf}\downarrow & & \downarrow{\scriptstyle \exists!} \\ TY & \xrightarrow{q_Y} & T'Y \end{array} \tag{3}$$

We define $T'f$ to be this uniquely induced map. Since $q_X \circ r_X = 1_{T'X}$, it follows that we may describe $T'f$ concretely as $T'f = q_Y \circ T(f) \circ r_X$. The uniqueness of $T'f$ means that the definition of $T'f$ does not depend on (the particular choice of representatives made by) r_X.

Definition 1 (Quotient monad). *Given a monad $\langle T, \eta, \mu\rangle$ and equations $E \subseteq TV \times TV$, we define the "quotient monad" $\langle T', \eta', \mu'\rangle$ as follows.*

$$\begin{aligned} T'X &= TX/{\equiv_X}, \\ T'(f\colon X \to Y) &= q_Y \circ T(f) \circ r_X, \\ \eta'_X &= q_X \circ \eta_X, \\ \mu'_X &= q_X \circ \mu_X \circ r_{TX} \circ T'(r_X) \end{aligned}$$

Proposition 1. *Given any monad $\langle T, \eta, \mu\rangle$, the quotient monad $\mathcal{T}' = \langle T', \eta', \mu'\rangle$ is indeed a monad, and $q\colon T \Rightarrow T'$ is a monad map.*

The above construction yields a concrete monad $\mathcal{T}'$ given a set of operations and equations. Intuitively, any monad which is isomorphic to $\mathcal{T}'$ is presented by these same operations and equations; this is captured by the following definition.

Definition 2. *Let Σ be an endofunctor, $\mathcal{T}$ the corresponding free monad, $E \subseteq TV \times TV$ a set of equations and $\mathcal{T}'$ the quotient monad. We say that a monad $\mathcal{K} = \langle K, \theta, \nu\rangle$ is* presented by Σ and E *if there is a monad map $i\colon T' \Rightarrow K$ which is a natural isomorphism.*

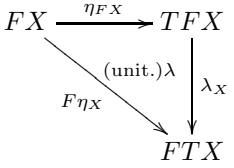
Example 1. The *idempotent semiring monad* is defined by the functor mapping a set X to the set $\mathcal{P}_\omega(X^*)$ of finite languages over X and, for morphisms $f\colon X \to Y$ in Set we define $\mathcal{P}_\omega(f^*)(L) = \bigcup\{f(x_1)\cdots f(x_n) \mid x_1 \cdots x_n \in L\}$. Further, $\eta_X\colon X \to \mathcal{P}_\omega(X^*)$ is given by $\eta_X(x) = \{x\}$ and $\mu_X\colon \mathcal{P}_\omega(\mathcal{P}_\omega(X^*)^*) \to \mathcal{P}_\omega(X^*)$ by $\mu_X(\mathcal{L}) = \bigcup_{L_1\cdots L_n \in \mathcal{L}}\{w_1 \cdots w_n \mid w_i \in L_i\}$. It is presented by two constants $\mathsf{0}$ and $\mathsf{1}$, two binary operations $+$ and $\cdot$, and the idempotent semiring axioms. The witnessing isomorphism can easily be given based on the observation that every semiring term is equivalent with respect to the idempotent semiring equations to a sum of products of variables.

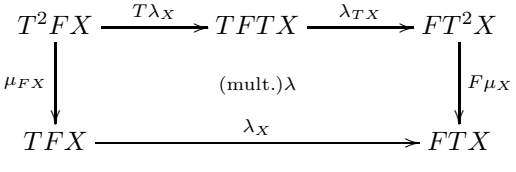
3 Distributive Laws and Bialgebras

We briefly recall the basic definitions of distributive laws and bialgebras; for a more thorough introduction we refer to [8,2,19].

3.1 Basic Definitions

Let $\mathcal{T} = \langle T, \eta, \mu \rangle$ be a finitary Set-monad, and F a Set-functor. A *distributive law* λ of the monad $\mathcal{T}$ over the functor F is a natural transformation $\lambda\colon TF \Rightarrow FT$ which is compatible with the monad structure, meaning that $\lambda \circ \eta_F = F\eta$ and $\lambda \circ \mu_F = F\mu \circ \lambda_T \circ T\lambda$, i.e., for all X the following diagrams commute:





We recall that every distributive law $\lambda\colon TF \Rightarrow FT$ corresponds to a *lifting* F_λ of F to the category of $\mathcal{T}$-algebras (see, e.g., [7,8]), defined as

$$F_\lambda\langle A, \alpha\rangle = \langle FA, F\alpha \circ \lambda_A\rangle \qquad F_\lambda(f) = Ff \tag{4}$$

Note that the compatibility of λ_X with μ_X means precisely that λ_X is a $\mathcal{T}$-algebra homomorphism from $\langle TFX, \mu_{FX}\rangle$ to $F_\lambda\langle TX, \mu_X\rangle$.

An *F-coalgebra* is a pair $\langle X, c\rangle$ where X is a set and $c\colon X \to FX$ is a map. An *F-coalgebra morphism* from $\langle X, c\rangle$ to $\langle Y, d\rangle$ is a map $f\colon X \to Y$ such that $d \circ f = Tf \circ c$. A *λ-bialgebra* is a triple $\langle X, \alpha, \beta\rangle$ where $\alpha\colon TX \to X$ is a $\mathcal{T}$-algebra and $\beta\colon X \to FX$ is an F-coalgebra such that $\beta \circ \alpha = F\alpha \circ \lambda_X \circ T\beta$. A *morphism of λ-bialgebras* from $\langle X_1, \alpha_1, \beta_1\rangle$ to $\langle X_2, \alpha_2, \beta_2\rangle$ is a function $f\colon X_1 \to X_2$ which is both a $\mathcal{T}$-algebra morphism and an F-coalgebra morphism.

The following results are well known (e.g., [2,8]). If $\langle Z, \zeta\rangle$ is a final F-coalgebra, then a distributive law $\lambda\colon TF \Rightarrow FT$ yields a final λ-bialgebra $\langle Z, \alpha, \zeta\rangle$ where $\alpha\colon TZ \to Z$ is defined by coinduction from the F-coalgebra $\langle TZ, \lambda_Z \circ T\zeta\rangle$.

We will need the notion of distributive laws of monads over *copointed* functors. A copointed functor is a pair $\langle F, \epsilon\rangle$ where F is an endofunctor and $\epsilon\colon F \Rightarrow \mathrm{Id}$ a natural transformation. A distributive law of $\mathcal{T}$ over $\langle F, \epsilon\rangle$ is a distributive law of $\mathcal{T}$ over F additionally satisfying $\epsilon_T \circ \lambda = T\epsilon$. For any Set-functor F, the *cofree copointed functor generated by* F is the pair $\langle \mathrm{Id} \times F, \pi_1\colon \mathrm{Id} \times F \to \mathrm{Id}\rangle$ where π_1 is the natural left-projection.

When $\mathcal{T}$ is the free monad generated by a signature functor Σ, then distributive laws involving $\mathcal{T}$ can be reduced to "plain" natural transformations using recursion, namely, there is a 1-1 correspondence between distributive laws $\lambda\colon TF \Rightarrow FT$ of $\mathcal{T}$ over F and natural transformations $\rho\colon \Sigma F \Rightarrow FT$ (cf. [2]). Such a ρ corresponds to a specification format of operational rules, and is sometimes referred to as a *simple SOS rule*. Similarly, for cofree copointed functors,

if $\mathcal{T}$ is freely generated by Σ, then there is a 1-1 correspondence between distributive laws $\lambda\colon T(\mathrm{Id}\times F)\Rightarrow(\mathrm{Id}\times F)T$ of $\mathcal{T}$ over $\langle \mathrm{Id}\times F,\pi_1\rangle$ and natural transformations $\rho\colon \Sigma(\mathrm{Id}\times F)\Rightarrow FT$ (cf. [10,4]). Such a natural transformation ρ is also referred to as an *abstract GSOS-rule* since it generalises the GSOS-format for labelled transition systems where $F=\mathcal{P}_\omega(-)^A$, cf. [2,19].

3.2 Solutions to Corecursive Equations

An important application of distributive laws is in solving *corecursive equations* which are maps of the type $\phi\colon X\to FTX$ where F is a functor and T is (the functor component of) a monad. These include many interesting and useful structures such as linear and context-free systems of behavioural differential equations [16,22], as well as linear, nondeterministic and weighted automata cf. [4,18]. These are all instances of $\mathcal{T}$*-automata* [4] which have the type $X\to B\times(TX)^A$ where A is a set and B carries a $\mathcal{T}$-algebra $\beta\colon TB\to B$, i.e., in particular, $F=B\times(-)^A$ whose final coalgebra carrier is B^{A^*}.

In the presence of a distributive law $\lambda\colon TF\Rightarrow FT$ one obtains a λ*-coinduction principle* [2] which provides unique solutions in the final λ-bialgebra $\langle Z,\alpha,\zeta\rangle$ to corecursive equations of the form $\phi\colon X\to FTX$. Ordinary coinduction is the special case where $\mathcal{T}$ is the identity monad. Formally, a solution to $\phi\colon X\to FTX$ in a λ-bialgebra $\langle A,\alpha,\beta\rangle$ is a map $f\colon X\to A$ such that

$$\begin{array}{ccccc} X & & \xrightarrow{\quad f\quad} & & A \\ {\scriptstyle\phi}\downarrow & & & & \downarrow{\scriptstyle\beta} \\ FTX & \xrightarrow{FTf} & FTA & \xrightarrow{F\alpha} & FA \end{array} \tag{5}$$

commutes. More precisely, λ-coinduction is coinduction in the category of λ-bialgebras, and we have the following fact.

Proposition 2 (Lemmas 4.3.3 and 4.3.4 of [2]). *Let $\phi\colon X\to FTX$ be a corecursive equation. Taking $\phi^\lambda=F\mu_X\circ\lambda_{TX}\circ T\phi$ then $\langle TX,\mu_X,\phi^\lambda\rangle$ is a λ-bialgebra, and $\eta_X\colon X\to TX$ is a solution of ϕ. Moreover, for any λ-bialgebra $\langle A,\alpha,\beta\rangle$, there is a 1-1 correspondence between solutions of ϕ in $\langle A,\alpha,\beta\rangle$ and λ-bialgebra morphisms from $\langle TX,\mu_X,\phi^\lambda\rangle$ to $\langle A,\alpha,\beta\rangle$.*

A "pointwise distributive law" λ for $\mathcal{T}$-automata can be obtained (cf. [4,6]) by taking $\lambda_X=(\beta\times\mathsf{st})\circ\langle T\pi_1,T\pi_2\rangle$ where $\mathsf{st}\colon T\circ(-)^A\Rightarrow(-)^A\circ T$ is the strength natural transformation. This λ is named so, since it induces the pointwise extension of $\beta\colon TB\to B$ on B^{A^*}. In the context-free and streams examples below, however, the desired algebraic structure on B^{A^*} uses the convolution product which is not the pointwise extension of the semiring product of B. So for these examples a different λ must be given.

4 Quotients of Distributive Laws

In Section 2 we saw how term equations give rise to quotients of algebras, and we gave an explicit construction of the resulting quotient monad. In this section,

we investigate conditions under which distributive laws and equations give rise to quotients of distributive laws.

As before, let Σ be a finitary signature functor generating the free monad $\mathcal{T} = \langle T, \eta, \mu \rangle$, and let E be a set of T-equations with associated T-congruence $\equiv_X$ on terms, and quotient monad $\mathcal{T}' = \langle T', \eta', \mu' \rangle$.

4.1 Distributive Laws over Plain Behaviour Functors

In this subsection, we assume that $\lambda\colon TF \Rightarrow FT$ is a distributive law of $\mathcal{T}$ over a plain behaviour functor F. We will provide a condition on λ and the equations E that ensures that we get a distributive law $\lambda'\colon T'F \Rightarrow FT'$ for the quotient monad. To this end, it is convenient to use the notion of a morphism of distributive laws from [20].

Definition 3. *Let $\langle T, \eta, \mu \rangle$ and $\langle K, \theta, \nu \rangle$ be monads, and let $\lambda\colon TF \Rightarrow FT$ and $\kappa\colon KF \Rightarrow FK$ be distributive laws. A natural transformation $\tau\colon T \Rightarrow K$ is a morphism from λ to κ (notation $\tau\colon \lambda \Rightarrow \kappa$) if τ is a monad morphism and the following square commutes:*

$$\begin{array}{ccc} TF & \xrightarrow{\tau F} & KF \\ {\scriptstyle\lambda}\downarrow & & \downarrow{\scriptstyle\kappa} \\ FT & \xrightarrow{F\tau} & FK \end{array} \tag{6}$$

We note that there are generalisations of the above definition that allow natural transformations between behaviour functors, cf. [20]. For our purposes, we do not need to change the behaviour type.

Definition 4. *We say that $\lambda\colon TF \Rightarrow FT$* preserves (equations in) *E if for all $g\colon V \to FX$, and for all $s, t \in TV$:*

$$s \; E \; t \quad \Rightarrow \quad Fq_X(\lambda_X(Tg(s))) \;=\; Fq_X(\lambda_X(Tg(t))). \tag{7}$$

Equation (7) can be conveniently formulated in terms of relation lifting as

$$s \; E \; t \quad \Rightarrow \quad \lambda_X(Tg(s)) \; \overline{F}(\equiv_X) \; \lambda_X(Tg(t)). \tag{8}$$

where the F-lifting of a relation $R \subseteq Y \times Y$ is defined as

$$\overline{F}(R) = \{\langle F\pi_1(u), F\pi_2(u) \rangle \in FY \times FY \mid u \in F(R)\}$$

and noticing that $u \; \overline{F}(\equiv_X) \; v$ iff $Fq_X(u) = Fq_X(v)$.

We can now state our main result.

Theorem 1. *The following are equivalent.*

1. *$\lambda\colon TF \Rightarrow FT$ preserves equations E.*
2. *there is a (unique) distributive law $\lambda'\colon T'F \Rightarrow FT'$ such that the quotient map $q\colon T \Rightarrow T'$ is a morphism of distributive laws from λ to λ'.*

Remark 1. Using that distributive laws correspond to functor liftings on $\mathcal{T}$-algebras (cf. (4)), the distributive law λ' in Theorem 1 exists if and only if the functor F_λ restricts to $\mathcal{T}'$-algebras. A similar statement for the case when F is a monad is made in [11, Corollary 3.4.2].

As a corollary we obtain the analogue of Theorem 1 for monads presented by operations and equations.

Corollary 1. *Suppose $\mathcal{K} = \langle K, \theta, \nu\rangle$ is presented by operations Σ and equations E with natural isomorphism $i\colon T' \Rightarrow K$, and suppose we have a distributive law $\lambda\colon TF \Rightarrow FT$ of $\mathcal{T}$ over F. Then there exists a unique distributive law $\kappa\colon KF \to FK$ of $\mathcal{K}$ over F such that $i \circ q\colon \lambda \Rightarrow \kappa$ is a morphism of distributive laws.*

Theorem 1 says that if λ preserves the equations in E, then we can *present* λ' *as "λ modulo equations"*. We illustrate this with an example.

Example 2 (Stream calculus). Behavioural differential equations are used extensively in [16,17] to define streams and stream operations. Here, the behaviour functor is $F(X) = \mathbb{R} \times X$ whose final coalgebra $\langle \mathbb{R}^\omega, \zeta\rangle$ consists of streams over the real numbers together with the map $\zeta(\sigma) = \langle \sigma(0), \sigma'\rangle$ which maps a stream σ to its initial value $\sigma(0)$ and derivative σ'.

The following behavioural specification defines the constant streams $[a] = (a, 0, 0, \ldots)$ for all $a \in \mathbb{R}$, $\mathtt{X} = (0, 1, 0, 0, \ldots)$, pointwise addition and convolution product of streams. We point out that the convolution product is defined here by a simple stream SOS-rule rather than a stream GSOS-rule, which is used in [16,17]. We explain this choice at the end of the example.

$$\begin{array}{rll}
[a](0) = a, & [a]' = [0], & \forall a \in \mathbb{R} \\
\mathtt{X}(0) = 0, & \mathtt{X}' = [1], & \\
(\sigma + \tau)(0) = \sigma(0) + \tau(0), & (\sigma + \tau)' = \sigma' + \tau', & \\
(\sigma \times \tau)(0) = \sigma(0) \cdot \tau(0), & (\sigma \times \tau)' = (\sigma' \times [\tau(0)]) + (\sigma' \times \mathtt{X} \times \tau') + ([\sigma(0)] \times \tau') &
\end{array}$$

The signature functor is thus $\Sigma(X) = \mathbb{R} + 1 + (X \times X) + (X \times X)$, and the above specification induces a distributive law $\lambda\colon TF \Rightarrow FT$. We would like to apply Theorem 1 to obtain a distributive law λ' for the quotient monad $\mathcal{T}'$ arising from $\mathcal{T}$ and E. Let E consist of the following axioms where $V = \{v, u, w\}$ and $a, b \in \mathbb{R}$:

$$\begin{array}{lll}
(v + u) + w = v + (u + w) & [0] + v = v & v + u = u + v \\
(v \times u) \times w = v \times (u \times w) & [1] \times v = v & v \times u = u \times v \\
v \times (u + w) = (v \times u) + (v \times w) & [0] \times v = [0] & \\
[a + b] = [a] + [b] & [a \cdot b] = [a] \times [b] &
\end{array} \tag{9}$$

E consists of the *commutative* semiring axioms together with axioms stating the inclusion of the underlying semiring of the reals. We show that λ preserves E. Let $g\colon V \to FX$ be arbitrary and suppose $g(v) = \langle a, x\rangle, g(u) = \langle b, y\rangle, g(z) = \langle c, z\rangle$. First note that for $F = \mathbb{R} \times \mathrm{Id}$, $\langle r_1, t_1\rangle\ \overline{F}(\equiv_X)\ \langle r_2, t_2\rangle$ iff $r_1 = r_2$ and $t_1 \equiv_X t_2$. It is straightforward to check preservation of the axioms that only concern addition,

as well as of $[1]\times v = v$, $[0]\times v = [0]$ and $v\times u = u\times v$. We show that $[a\cdot b] = [a]\times[b]$ is preserved:

$$\begin{array}{rcl}\lambda_X([a]\times[b]) & = & \langle a\cdot b, [0]\times[b] + [0]\times \mathtt{X} \times [0] + [a]\times[0]\rangle \\ \overline{F}(\equiv_X)\ \langle a\cdot b, [0]\rangle & = & \lambda_X([a\cdot b])\end{array}$$

We check that λ preserves the distribution axiom:

$$\begin{array}{l}\lambda_X(\langle a, x\rangle \times (\langle b, y\rangle + \langle c, z\rangle)) \\ \quad = \quad \langle a\cdot(b+c), (x\times[b+c]) + (x\times X\times(y+z)) + [a]\times(y+z)\rangle \\ \quad \overline{F}(\equiv_X)\ \langle a\cdot(b+c), (x\times[b+c]) + (x\times X\times y) + (x\times X\times z) + \\ \qquad\qquad ([a]\times y) + ([a]\times z)\rangle \\ \quad \overline{F}(\equiv_X)\ \langle (a\cdot c) + (b\cdot c), (x\times[b]) + (x\times X\times y) + ([a]\times y) + \\ \qquad\qquad (x\times[c]) + (x\times X\times z) + ([a]\times z)\rangle \\ \quad = \\ \lambda_X((\langle a, x\rangle \times \langle b, y\rangle) + (\langle a, x\rangle \times \langle c, z\rangle))\end{array}$$

Note that we used $[a + b] = [a] + [b]$. Similarly, preservation of $\times$-associativity can be verified, and it uses the axiom $[a\cdot b] = [a]\times[b]$. We have thus shown that λ preserves E, and it follows, in particular, that $\langle \mathbb{R}^\omega, +, \times, [0], [1]\rangle$ is a commutative semiring. This was shown directly in [17], but the proof uses bisimulation-up-to as well as the fundamental theorem of stream calculus, which cannot be added as an equation. In our approach we construct a distributive law, and obtain not only this result but also the soundness of the bisimulation-up-to technique [15], and the existence of unique solutions to corecursive equations $\phi\colon X \to FT'X$ (see Section 3.2).

The derivative of the convolution product is usually (cf. [16,17]) specified as:

$$(\sigma\times\tau)' = (\sigma'\times\tau) + ([\sigma(0)]\times\tau') \tag{10}$$

which corresponds to a stream GSOS-rule $\Sigma(\mathrm{Id}\times\mathbb{R}\times\mathrm{Id}) \Rightarrow \mathbb{R}\times T(-)$, and thus to a distributive law over the cofree copointed functor. However, with this definition, we could not show that the commutativity of $\times$ is preserved although all other axioms remain preserved. Hence a given λ does not necessarily satisfy all equations that are valid on the final F-coalgebra.

In the above example, we did not have a concrete monad in mind; we simply considered a free monad and a set of equations. In Example 4 below we give an example for the concrete idempotent semirings monad.

Remark 2. The concrete proof method for preservation of equations bears a close resemblance to *bisimulation up to congruence* [15], in that one must show that for every pair in E_{FX} its derivatives are related by the least congruence $\equiv_X$ instead of E_X.

Example 3. We have seen in the discussion of (10) that equations that hold in the final coalgebra are not necessarily preserved by λ. Now we give another concrete example of this fact. This example again concerns stream systems, i.e.,

coalgebras for the functor $FX = \mathbb{R} \times X$. We define the constant stream of zeros by three different constants n_1, n_2 and n_3 by the following behavioural differential equations:

$$n_1(0) = 0,\ n_1' = n_1 \qquad n_2(0) = 0,\ n_2' = n_3 \qquad n_3(0) = 0,\ n_3' = n_3$$

The corresponding signature functor is thus $\Sigma X = 1+1+1$, and the above specification gives rise to a distributive law $\lambda\colon TF \Rightarrow FT$ where T is (the functorial component of) the free monad over Σ. Now consider the equation $n_1 = n_2$; this clearly holds when interpreted in the final coalgebra. However, this equation is not preserved by λ. To see this, notice that $\lambda(n_1) = \langle 0, n_1\rangle$ and $\lambda(n_2) = \langle 0, n_3\rangle$, but $n_1 \not\equiv_X n_3$, so $\lambda(n_1)$ and $\lambda(n_2)$ are not related by $\overline{F}(\equiv_X)$.

4.2 Distributive Laws over Copointed Functors

We now show that our main results hold as well for distributive laws of monads over *copointed* functors. This extends our method to deal with operations specified in abstract GSOS format, such as language concatenation.

Proposition 3. *Theorem 1 and Corollary 1 hold as well for any distributive law of a monad over a copointed functor.*

Example 4 (Context-free languages). A context free grammar (in Greibach normal form) consists of a finite set A of terminal symbols, a (finite) set X of non-terminal symbols, and a map $\langle o, t\rangle\colon X \to 2 \times \mathcal{P}_\omega(X^*)^A$, i.e., it is a coalgebra for the behavior functor $F = 2\times \mathrm{Id}^A$ composed with the idempotent semiring monad $\mathcal{P}_\omega(\mathrm{Id}^*)$ from Example 1. Intuitively, $o(x) = 1$ means that the variable x can generate the empty word, whereas $w \in t(x)(a)$ if and only if x can generate aw, cf. [22].

It is a rather difficult task to describe concretely a distributive law of $\mathcal{P}_\omega(\mathrm{Id}^*)$ over F (or $\mathrm{Id} \times F$) defining the sum $+$ and sequential composition $\cdot$ of context-free grammars. More conveniently, since we have seen in Example 1 that the monad $\mathcal{P}_\omega(\mathrm{Id}^*)$ can be presented by the operations and axioms of idempotent semirings, we proceed by defining a distributive law λ of the free monad $\mathcal{T}$ generated by the semiring signature functor $\Sigma(X) = 1+1+(X\times X)+(X\times X)$ over the cofree copointed functor $\langle \mathrm{Id}\times F, \pi_1\rangle$, and show that λ preserves the semiring axioms. We define λ as the distributive law that corresponds to the natural transformation $\rho\colon \Sigma(\mathrm{Id}\times F) \Rightarrow FT$ whose components are given by:

$$\begin{aligned}
\rho^0_X &= \langle 0, a \mapsto \emptyset\rangle \\
\rho^1_X &= \langle 1, a \mapsto \emptyset\rangle \\
\rho^+_X(\langle x, o, f\rangle, \langle y, p, g\rangle) &= \langle \max\{o,p\}, a \mapsto f(a) + g(a)\rangle \\
\rho^{\cdot}_X(\langle x, o, f\rangle, \langle y, p, g\rangle) &= \left\langle \min\{o,p\}, a \mapsto \begin{cases} f(a)\cdot y & \text{if } p = 0 \\ f(a)\cdot y + g(a) & \text{if } p = 1 \end{cases} \right\rangle
\end{aligned} \tag{11}$$

We proceed to show that λ preserves the defining equations of idempotent semirings. We treat here only the case of distributivity, i.e., $u\cdot(v+w) = u\cdot v + u\cdot w$. To

this end let $g\colon V \to X \times FX$ be arbitrary and suppose $g(v) = \langle x, o, d\rangle, g(u) = \langle y, p, e\rangle$ and $g(z) = \langle z, q, f\rangle$. Notice that either $o = 0$ or $o = 1$; we treat both cases separately:

$$\begin{aligned}
&\lambda(\langle x,0,d\rangle \cdot (\langle y,p,e\rangle + \langle z,q,f\rangle)) \\
&\quad = \quad (x \cdot (y+z), 0, a \mapsto d(a) \cdot (y+z)) \\
&\quad \overline{F}(\equiv_X)\ (x \cdot y + x \cdot z, 0, a \mapsto d(a) \cdot y + d(a) \cdot z) \\
&\quad = \quad \lambda(\langle x,0,d\rangle \cdot \langle y,p,e\rangle + \langle x,0,d\rangle \cdot \langle z,q,f\rangle)
\end{aligned}$$

$$\begin{aligned}
&\lambda(\langle x,1,d\rangle \cdot (\langle y,p,e\rangle + \langle z,q,f\rangle)) \\
&\quad = \quad (x \cdot (y+z), p+q, a \mapsto d(a) \cdot (y+z) + (e(a) + f(a))) \\
&\quad \overline{F}(\equiv_X)\ (x \cdot y + x \cdot z, p+q, a \mapsto (d(a) \cdot y + d(a) \cdot z) + (e(a) + f(a))) \\
&\quad \overline{F}(\equiv_X)\ (x \cdot y + x \cdot z, p+q, a \mapsto (d(a) \cdot y + e(a)) + (d(a) \cdot z + f(a))) \\
&\quad = \quad \lambda(\langle x,1,d\rangle \cdot \langle y,p,e\rangle + \langle x,1,d\rangle \cdot \langle z,q,f\rangle)\,.
\end{aligned}$$

In a similar way one can show that λ preserves the other idempotent semiring equations. Thus, from Proposition 3 and Corollary 1 we obtain a distributive law κ of $\mathcal{P}_\omega(\mathrm{Id}^*)$ over $2 \times \mathrm{Id}^A$ such that $i \circ q\colon \lambda \Rightarrow \kappa$ is a morphism of distributive laws, i.e., κ is presented by λ and the equations of idempotent semirings.

5 Morphisms and Solutions

In this section, we show that morphisms of distributive laws commute with solving corecursive equations. In the case of monads with equations, this means that first solving equations ϕ with respect to $\mathcal{T}$ and then quotienting the solution bialgebra is the same as first quotienting $\mathcal{T}$ and solving with respect to the quotient monad $\mathcal{T}'$.

We first describe some functors that link the relevant categories of bialgebras and corecursive equations. Throughout this Section, we let $\mathcal{T} = \langle T, \eta, \mu\rangle$ and $\mathcal{K} = \langle K, \theta, \nu\rangle$ be monads; and $\lambda\colon TF \Rightarrow FT$ and $\kappa\colon KF \Rightarrow FK$ be distributive laws of $\mathcal{T}$ and $\mathcal{K}$ over F, respectively.

If $\tau\colon \lambda \Rightarrow \kappa$ is a morphism of distributive laws, then precomposing with τ yields a functor:

$$\begin{array}{lccc}
I\colon & \mathsf{Bialg}(\kappa) & \to & \mathsf{Bialg}(\lambda) \\
KX \xrightarrow{\alpha} X \xrightarrow{\beta} FX & \mapsto & TX \xrightarrow{\alpha\circ\tau_X} X \xrightarrow{\beta} FX
\end{array} \tag{12}$$

I takes a κ-bialgebra to a λ-bialgebra follows from the naturality of τ and $F\tau \circ \lambda = \kappa \circ \tau F$. Similarly, postcomposing with $F\tau$ yields a functor between corecursive equations:

$$\begin{array}{lccc}
Q\colon & \mathsf{Coalg}(FT) & \to & \mathsf{Coalg}(FK) \\
\phi\colon X \to FTX & & \mapsto & F\tau_X \circ \phi\colon X \to FKX
\end{array} \tag{13}$$

Recall from Section 3.2, that given a distributive law $\lambda\colon TF \Rightarrow FT$, the solutions of a corecursive equation $\phi\colon X \to FTX$ are characterised by morphisms from the λ-bialgebra $\langle TX, \mu_X, \phi^\lambda\rangle$ whose F-coalgebra structure given by

$$\phi^\lambda = F\mu_X \circ \lambda_{TX} \circ T\phi \tag{14}$$

This yields a functor (see e.g. [5, Lem. 5.4.11]):

$$\begin{array}{rccc} G_\lambda\colon & \mathsf{Coalg}(FT) & \to & \mathsf{Bialg}(\lambda) \\ & \langle X, \phi\rangle & \mapsto & \langle TX, \mu_X, \phi^\lambda\rangle \end{array} \tag{15}$$

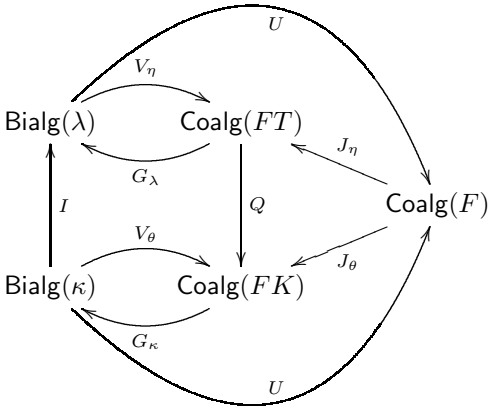
We can go in the opposite direction by using the monad unit,

$$\begin{array}{rccc} V_\eta\colon & \mathsf{Bialg}(\lambda) & \to & \mathsf{Coalg}(FT) \\ & \langle X, \alpha, \beta\rangle & \mapsto & \langle X, F\eta_X \circ \beta\rangle \end{array} \tag{16}$$

which decomposes into the functor $U\colon \mathsf{Bialg}(\lambda) \to \mathsf{Coalg}(F)$ that forgets algebra structure, and

$$\begin{array}{rccc} J_\eta\colon & \mathsf{Coalg}(F) & \to & \mathsf{Coalg}(FT) \\ & \langle X, \beta\rangle & \mapsto & \langle X, F\eta_X \circ \beta\rangle \end{array} \tag{17}$$

The following diagram summarises the situation:



(18)

We mention that $QV_\eta I = V_\theta$ since τ is compatible with the units of $\mathcal{T}$ and $\mathcal{K}$.

Morphisms of distributive laws are defined to be monad maps, and hence respect the algebraic structure. The next proposition shows that, as one might expect, they also respect the coalgebraic structure, and hence morphisms of distributive laws induce morphisms between bialgebras.

Proposition 4. *If $\tau\colon \lambda \Rightarrow \kappa$ is a morphism of distributive laws, then for all $\phi : X \to FTX$ we have that τ_X is a λ-bialgebra morphism $\tau_X\colon G_\lambda(\phi) \to IG_\kappa Q(\phi)$ or, equivalently, an F-coalgebra morphism $\tau_X\colon \langle TX, \phi^\lambda\rangle \to \langle KX, (Q\phi)^\kappa\rangle$.*

It follows that the unique λ-bialgebra morphism $g\colon \langle TX, \mu_X, \phi^\lambda\rangle \to \langle Z, \alpha, \zeta\rangle$ into the final λ-bialgebra $\langle Z, \alpha, \zeta\rangle$ factors as $g = g' \circ \tau_X$, where g' is the final λ-bialgebra morphism from $IG_\kappa Q(\phi)$, as shown here:

$$\begin{array}{ccccccc}
T^2X & \xrightarrow{T\tau_X} & TKX & \xrightarrow{Tg'} & TZ \\
\downarrow{\scriptstyle \mu_X} & & \downarrow{\scriptstyle \nu_X \circ \tau_{KX}} & & \downarrow{\scriptstyle \alpha} \\
X \xrightarrow{\eta_X} TX & \xrightarrow{\tau_X} & KX & \xrightarrow{g'} & Z \\
\searrow^{\phi} \downarrow{\scriptstyle \phi^\lambda} & & \downarrow{\scriptstyle (Q\phi)^\kappa} & & \downarrow{\scriptstyle \zeta} \\
FTX & \xrightarrow{F\tau_X} & FKX & \xrightarrow{Fg'} & FZ
\end{array} \tag{19}$$

Hence by Proposition 2, every solution of ϕ in the final λ-bialgebra yields a solution of $Q\phi$, and vice versa.

When $\tau\colon \lambda \Rightarrow \kappa$ arises from a set of preserved equations E as in Section 4 (with $\kappa = \lambda'$), then Proposition 4 says that $IG_\kappa Q(\phi)$ is a quotient of the "free" λ-bialgebra $\langle TX, \mu_X, \phi^\lambda \rangle$, and in particular, the congruence $\equiv_X$ is an F-behavioural equivalence. In this case, $Q\phi$ is the corecursive equation obtained by reading the right-hand side of ϕ modulo equations in E. In other words, quotienting the solution of the equation ϕ is the same as solving the quotiented equation $Q\phi$.

Example 5. Recall from Example 4 that $i \circ q\colon T \Rightarrow \mathcal{P}_\omega(X^*)$ is a morphism of distributive laws. By Proposition 4 we have the following commuting diagram for any corecursive equation $\phi\colon X \to 2 \times (TX)^A$:

$$\begin{array}{ccccccc}
X & \xrightarrow{\eta_X} & TX & \xrightarrow{i \circ q_X} & \mathcal{P}_\omega(X^*) & \longrightarrow & \mathcal{P}(A^*) \\
\searrow^{\phi} & & \downarrow{\scriptstyle \phi^\lambda} & & \downarrow{\scriptstyle (Q\phi)^\kappa} & & \downarrow{\scriptstyle \zeta} \\
 & & 2 \times (TX)^A & \xrightarrow{1 \times (i \circ q_X)^A} & 2 \times (\mathcal{P}_\omega(X^*))^A & \longrightarrow & 2 \times \mathcal{P}(A^*)^A
\end{array} \tag{20}$$

Notice that a context-free grammar $\langle o, t \rangle\colon X \to 2 \times \mathcal{P}_\omega(X^*)^A$ can be represented by a $\phi\colon X \to 2 \times (TX)^A$ such that $Q\phi = \langle o, t \rangle$, since $i \circ q$ is surjective. This gives the expected correspondence between two of the three different coalgebraic approaches to context-free languages introduced in [22] (the third approach is about fixed-point expressions and as such is outside the scope of this paper).

Similarly, the algebraic structure induced by λ on the final F-coalgebra factors uniquely through the algebraic structure induced by κ.

Proposition 5. *Let $\tau\colon \lambda \Rightarrow \kappa$ be a morphism of distributive laws, and let $\alpha\colon TZ \to Z$ and $\alpha'\colon KZ \to Z$ be the algebras induced by λ and κ respectively on the final coalgebra $\langle Z, \zeta \rangle$. Then $\alpha = \alpha' \circ \tau_Z$.*

Example 6. Continuing Example 5, it follows from Proposition 5 that the algebra $\alpha\colon T\mathcal{P}(A^*) \to \mathcal{P}(A^*)$ induced by the distributive law for $\mathcal{T}$ can be decomposed as $i \circ q \circ \alpha'$, where α' is the algebra on $\mathcal{P}(A^*)$ induced by the distributive law for $\mathcal{P}_\omega(\mathrm{Id}^*)$. It can be shown by induction that α is the algebra on languages given by union and concatenation product. Now $\alpha'\colon \mathcal{P}_\omega(\mathcal{P}(A^*)^*) \to \mathcal{P}(A^*)$ can be given by selecting a representative term and applying α, and it follows that $\alpha'(\mathcal{L}) = \bigcup_{L_1 \cdots L_n \in \mathcal{L}} \{w_1 \cdots w_n \mid w_i \in L_i\}$.

6 Discussion and Conclusion

We have presented a preservation condition that is necessary and sufficient for the existence of a distributive law λ' for a monad with equations given a distributive law λ for the underlying free monad. This condition consists of checking that the base equations are preserved by λ. For concrete monads, checking preservation is often much easier than describing and verifying the distributive law requirements directly. We demonstrated our method by applying it to obtain distributive laws for stream calculus over commutative semirings, and for context-free grammars which use the monad of idempotent semirings.

In [20] the notion of morphisms of distributive laws is studied as a general approach to translations between operational semantics. In this paper we investigate in detail the case of quotients of distributive laws. Distributive laws for monad quotients and equations are also studied in [10,11]. The setting and motivation of [11] is different as they study distributive laws of one monad over another with the aim to compose these monads. We study distributive laws of a monad over a plain or copointed functor. The approach in [10] is more general as they consider monads on arbitrary categories, but it also differs from ours in that the desired distributive law is contingent on two given distributive laws and the existence of the coequaliser (in the category of monads) which encodes equations. We have given a more direct analysis for monads in Set and a practical proof principle, which covers many known examples. We leave as future work to find out precisely how their Theorem 31 relates to our Theorem 1.

While in this work we have focused on adding equations which already hold in the final bialgebra, it is often useful to use equations to *induce* behaviour, next to a behavioural specification in terms of a distributive law. In process theory this idea is captured by the notion of structural congruences [13]; at the more general level of distributive laws there is work on adding recursive equations [9]. We leave a general study of structural congruences for distributive laws, and the precise relation to the present results, as future work.

More technically, it remains an open problem whether a converse of Proposition 4 holds. Finally, as mentioned by one of the referees, it would be desirable to be able construct the quotient monad without the assumption of right-inverses by using a more categorical treatment of congruences as spans. Another abstract formulation of our results in terms of right Kan extensions has also been suggested. We intend to investigate these matters in future work.

Acknowledgements. We thank Neil Ghani, Bart Jacobs, Jan Rutten and Joost Winter for helpful discussions and suggestions.

References

1. Aceto, L., Fokkink, W., Verhoef, C.: Structural operational semantics. In: Bergstra, J., Ponse, A., Smolka, S. (eds.) Handbook of Process Algebra, pp. 197–292. Elsevier (2001)

2. Bartels, F.: On Generalised Coinduction and Probabilistic Specification Formats. Ph.D. thesis, Vrije Universiteit Amsterdam (2004)
3. Hansen, H., Klin, B.: Pointwise extensions of GSOS-defined operations. Math. Struct. in Comp. Sci. 21, 321–361 (2011)
4. Jacobs, B.: A bialgebraic review of deterministic automata, regular expressions and languages. In: Futatsugi, K., Jouannaud, J.-P., Meseguer, J. (eds.) Algebra, Meaning, and Computation. LNCS, vol. 4060, pp. 375–404. Springer, Heidelberg (2006)
5. Jacobs, B.: Introduction to coalgebra. towards mathematics of states and observations. version 2.0 (2012) (unpublished book draft)
6. Jacobs, B.: Distributive laws for the coinductive solution of recursive equations. Inf. Comput. 204(4), 561–587 (2006)
7. Johnstone, P.: Adjoint lifting theorems for categories of algebras. Bull. London Math. Society 7, 294–297 (1975)
8. Klin, B.: Bialgebras for structural operational semantics: An introduction. Theor. Comp. Sci. 412, 5043–5069 (2011)
9. Klin, B.: Adding recursive constructs to bialgebraic semantics. J. Logic and Algebraic Programming 60-61, 259–286 (2004)
10. Lenisa, M., Power, J., Watanabe, H.: Category theory for operational semantics. Theor. Comp. Sci. 327(1-2), 135–154 (2004)
11. Manes, E., Mulry, P.: Monad compositions I: General constructions and recursive distributive laws. Theory and Applications of Categories 18(7), 172–208 (2007)
12. Milius, S.: A sound and complete calculus for finite stream circuits. In: Proceedings of LICS 2012, pp. 421–430. IEEE Computer Society (2010)
13. Mousavi, M.R., Reniers, M.A.: Congruence for structural congruences. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 47–62. Springer, Heidelberg (2005)
14. Rot, J., Bonchi, F., Bonsangue, M., Pous, D., Rutten, J., Silva, A.: Enhanced coalgebraic bisimulation, `http://www.liacs.nl/~jrot/papers/up-to.pdf`
15. Rot, J., Bonsangue, M., Rutten, J.: Coalgebraic bisimulation-up-to. In: van Emde Boas, P., Groen, F.C.A., Italiano, G.F., Nawrocki, J., Sack, H. (eds.) SOFSEM 2013. LNCS, vol. 7741, pp. 369–381. Springer, Heidelberg (2013)
16. Rutten, J.: Behavioural differential equations: a coinductive calculus of streams, automata and power series. Theor. Comp. Sci. 308(1), 1–53 (2003)
17. Rutten, J.: A coinductive calculus of streams. Math. Struct. in Comp. Sci. 15, 93–147 (2005)
18. Silva, A., Bonchi, F., Bonsangue, M., Rutten, J.: Generalizing the powerset construction, coalgebraically. In: Lodaya, K., Mahajan, M. (eds.) FSTTCS. LIPIcs, vol. 8, pp. 272–283. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2010)
19. Turi, D., Plotkin, G.: Towards a mathemathical operational semantics. In: Proceedings of LICS 1997, pp. 280–291. IEEE Computer Society (1997)
20. Watanabe, H.: Well-behaved translations between structural operational semantics. In: Moss, L. (ed.) Proceedings of CMCS 2002. ENTCS, vol. 65, pp. 337–357. Elsevier (2002)
21. Winskel, G.: The formal semantics of programming languages - an introduction. Foundation of computing series. MIT Press (1993)
22. Winter, J., Bonsangue, M.M., Rutten, J.: Context-free languages, coalgebraically. In: Corradini, A., Klin, B., Cîrstea, C. (eds.) CALCO 2011. LNCS, vol. 6859, pp. 359–376. Springer, Heidelberg (2011)

Interaction and Observation: Categorical Semantics of Reactive Systems Trough Dialgebras

Vincenzo Ciancia*

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

Abstract. We use dialgebras, generalising both algebras and coalgebras, as a complement of the standard coalgebraic framework, aimed at describing the semantics of an interactive system by the means of reaction rules. In this model, interaction is built-in, and semantic equivalence arises from it, instead of being determined by a (possibly difficult) understanding of the side effects of a component in isolation. Behavioural equivalence in dialgebras is determined by how a given process interacts with the others, and the obtained observations. We develop a technique to inter-define categories of dialgebras of different functors, that in particular permits us to compare a standard coalgebraic semantics and its dialgebraic counterpart. We exemplify the framework using the CCS and the π-calculus. Remarkably, the dialgebra giving semantics to the π-calculus does not require the use of presheaf categories.

1 Introduction

A system is called *interactive* when its semantics depends upon interaction with a surrounding environment. The semantics does not just yield a value (or not at all), but rather it consists in the denotation of the *behaviour* of the system itself, usually described either by *reaction rules* or by a *labelled transition system* (LTS). The difference is illustrated by the following example, defining a reaction rule for the synchronisation of two parallel processes in a process calculus (the rule on the left) or the LTS variant (the three rules on the right):

$$a.P \parallel \bar{a}.Q \to P \parallel Q \qquad a.P \xrightarrow{a} P \quad \bar{a}.P \xrightarrow{\bar{a}} P \quad \frac{P \xrightarrow{a} P' \;\; Q \xrightarrow{\bar{a}} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$$

Here $a.P$ is a process waiting for a signal on channel a, whose continuation is P. Similarly, $\bar{a}.P$ sends a signal on a, while $P \parallel Q$ is the parallel composition of

* The research leading to these results was partially supported by the EU FET 7FP Collaborative Project n. 600708 (QUANTICOL), the PAR FAS 2007-2013 (TRACE-IT) project, and the EU 7FP projects n. 256980 (NESSoS), n. 257930 (Aniketos), n. 295354 (SESAMO).

R. Heckel and S. Milius (Eds.): CALCO 2013, LNCS 8089, pp. 110–125, 2013.

two processes. The reaction rule may be read as "whenever two processes can synchronise, they do, and evolve into the parallel composition of their continuations". The LTS rules may be read as: "whenever a process can send or receive a signal, it evolves into its continuation, and has a *side effect* on the environment". Two processes with complementing side effects interact by the last rule.

LTSs are widely used for modelling the semantics of interactive systems, since they come equipped with bisimilarity, a form of *behavioural equivalence* that specifies when the semantics of two processes is the same. Coalgebras generalise LTSs and have a standard definition of bisimilarity, coinciding with kernel equivalence of morphisms under mild assumptions. Many formalisms received a coalgebraic treatment. This becomes increasingly harder as the complexity of the calculus grows, depending on the general question of *what are side effects* in a specific calculus. For example, dealing with name allocation in the π-calculus requires the use of presheaf categories (see e.g. [1]).

In this work, we seek for a setting where reaction rules are the main object of study, and side effects or labels are not needed at all to define behavioural equivalence. We use a generalisation of coalgebras, namely *dialgebras*, to represent a rule system as an object in a category, so that kernel equivalence can be used as a notion of behavioural equivalence. To appreciate the difference, consider a function $f : X \to \mathcal{P}(L \times X)$. It specifies an LTS with states in X describing, for each state x, the non-deterministic choices at x, the side effects of each choice, and the resulting state. In LTSs, and coalgebras, elements are observed in isolation. In contrast, an example of a dialgebra is $f : X \times X \to \mathcal{P}(X)$. The value of $f(x, y)$ is meant to describe the possible (non-deterministic) outcomes of an interaction between x and y. Elements are not observed in isolation, but rather their mutual interactions define the semantics.

Categories of dialgebras were first studied under the name of *generalised algebraic categories* (see e.g. [2,3]). These structures have been used in computer science for the specification of data types [4,5]. A systematic study of dialgebras, patterned after [6], was done in [7]. Applications to a compositional calculus for software components were proposed in [8]. In [9], we modelled asynchronous process calculi as isolated machines that can be fed with input tokens by an external observer, using dialgebras to generalise *Mealy machines*. In this paper we aim at a more intensional characterisation, tailored to reaction rules, obtained by studying interaction between pairs of systems rather than the relation between their input and output. Our work diverts from previous research, as we take a "local" approach. Instead of studying a whole category of dialgebras, one just considers objects that are reachable by morphisms from the particular dialgebra being studied. This approach addresses the issue, previously unsolved, that a final dialgebra fails to exist, so there is no universal semantic domain for the whole class of considered objects. To obviate to this, in §2, we study the *bisimilarity quotient* of a specific system, showing that it exists under mild conditions. The bisimilarity quotient is sufficient to provide a canonical semantics to an interactive system up-to behavioural equivalence, by the means of a canonical epimorphism. Indeed, a universal model is also useful to compare different

systems of the same type. However, this is not typical in coalgebraic program semantics, as different systems have different types, representing different kinds of side effects. In dialgebraic program semantics, just like in the coalgebraic case, working with different systems requires the use of categorical comparisons. In §4, we develop techniques to do so in the local perspective.

In §5 we provide two examples: the *Calculus of Communicating Systems* and the π-calculus. The simplicity of the former allows us to cleanly illustrate our framework, including the comparisons of §4, used to prove the equivalence of the coalgebraic and dialgebraic semantics. On the other hand, the dialgebraic semantics of the π-calculus provides an important insight: using reaction rules, one does not need to understand what are the side effects of programs, as interaction is sufficient to characterise their behaviour. Thus, the semantics of the calculus is given as a dialgebra in `Set`, without the need to resort to presheaf categories to describe bisimilarity; but nevertheless, we are able to prove that standard dialgebraic bisimilarity coincides with early bisimilarity. It is also worth notice that the close relationship between the two calculi permits us to define the dialgebra for the π-calculus by just changing one rule from the dialgebra for the CCS. This is possible since both systems can be described by the means of binary *interactions* and non-deterministic *observations*; that is, they use the same functors. We introduce these two simple functors in §3, together with a characterisation theorem for the associated behavioural equivalence.

Related Work. The most widely known framework for the categorical semantics of reactive systems is the so-called "*contexts as labels*" approach [10,11,12]. Roughly, an LTS is derived from reaction rules adopting unary contexts as labels; bisimilarity of such LTS serves as behavioural equivalence. Nevertheless, *minimal* contexts need to be carefully selected in order to obtain a sensible equivalence relation; the obtained semantics depends on this choice, and is not directly specified by the reaction rules themselves. The resulting categorical framework is highly non-trivial. Dialgebras, in contrast, provide a simple setting in which to study reactive systems. However, compositionality, which is a foundational reason to use contexts as labels, and certainly a desired feature, has not yet been investigated for dialgebras (more on this in the conclusions). For the time being, dialgebras complement algebras and coalgebras, rather than *bialgebras*.

2 Dialgebras

A coalgebraic semantics can be interpreted as the discerning power of an *observer* that can see all the actions done by a process. In contrast, *dialgebras* endow the observer with the ability to *interact* with the system. The passive observer becomes an entity which runs *experiments* and *observes* the results. By this change of point of view, we can represent e.g. input as an experiment in which the observer feeds a system with a value [9], or we can represent interaction as a binary experiment involving two processes as we do in the current paper.

We restrict all our definitions to the category `Set`, but indeed the theory of dialgebras can be developed in any category.

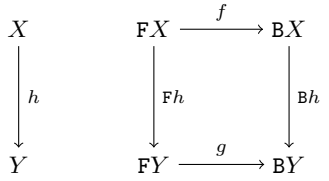


Fig. 1. A dialgebra homomorphism

Definition 1. *Given two functors* $\mathtt{F}, \mathtt{B} : \mathtt{Set} \to \mathtt{Set}$*, a* $(\mathtt{F}, \mathtt{B})$-dialgebra *is a pair* (X, f) *where* X *is the* carrier set *or* underlying set *and* $f : \mathtt{F}X \to \mathtt{B}X$ *is a function. A* $(\mathtt{F}, \mathtt{B})$*-dialgebra homomorphism from* (X, f) *to* (Y, g) *is a function* $h : X \to Y$ *such that* $g \circ \mathtt{F}h = \mathtt{B}h \circ f$*, as depicted in Figure 1. Dialgebras and their homomorphisms form the category* $Dialg(\mathtt{F}, \mathtt{B})$*.*

Roughly, $\mathtt{F}$ is the syntax of experiments, of which the function f is the semantics, yielding a set of elements in a type of observed results $\mathtt{B}$. The crucial feature of dialgebras is that, since they form a category, they have a standard notion of equivalence coming from kernels of morphisms. Notice that dialgebras conservatively extend algebras ($Dialg(\mathtt{F}, \mathtt{Id})$) and coalgebras ($Dialg(\mathtt{Id}, \mathtt{T})$).

Definition 2. *Given a* $(\mathtt{F}, \mathtt{B})$*-dialgebra* (X, f)*,* dialgebraic bisimilarity $\sim_f \subseteq X \times X$ *is defined by* $x \sim_f y \iff \exists (Y, g). \exists h : (X, f) \to (Y, g). h(x) = h(y)$*.*

Notice that in Definition 2 we use the kernel of h as a function. Thus, we do not require kernels (pullbacks of a function with itself) in $Dialg(\mathtt{F}, \mathtt{B})$. The "extensional" definition that we provide is applicable to any kind of dialgebra (independently from $\mathtt{F}$ and $\mathtt{B}$), and it avoids the machinery of relation liftings (which are used in [5]). We now study epi-mono factorisations of dialgebras.

Proposition 1. *If* $\mathtt{F}$ *and* $\mathtt{B}$ *preserve monos whose domain is empty, the category* $Dialg(\mathtt{F}, \mathtt{B})$ *has unique epi-mono factorisations. Otherwise, it has epi-mono factorisations of morphisms whose domain does not have an empty carrier.*

Proposition 1 guarantees that bisimilarity is determined by the epimorphisms. In coalgebras, the kernel of the unique morphism into the final object (if any) coincides with bisimilarity. For simple functors $\mathtt{F}$, e.g., $\mathtt{F}(X) = X \times X$, even when $\mathtt{B}$ is bounded, a final dialgebra does not exist[1].

Example 1. Let $\mathtt{F}(X) = X \times X$ and $\mathtt{B}(X) = \mathcal{P}_{fin}(X)$ (the finite power set of X). Suppose there is a final dialgebra (Z, z). Consider the dialgebra $(Z + 1, f)$ where $f(x, y) = z(x, y)$ if $x, y \in Z$, $f(x, *) = f(*, x) = \{*\}$ if $x \in Z$, and $f(*, *) = \emptyset$. Consider the final map $h : (Z + 1, f) \to (Z, f)$. Here we get to a contradiction:

[1] A final dialgebra still exists when $\mathtt{B}$ preserves the terminal object [7]. However, this makes the category of dialgebras not very interesting, as the final dialgebra has just one element, thus all the elements of any system are bisimilar.

h restricted to Z must be the identity, therefore injective and surjective. To see this, consider that (Z, z) embeds into $(Z + 1, f)$ by the identity function, and that the identity of (Z, z) must factor through such embedding and h, by finality. Thus, there is $x \in Z$ such that $h(x) = h(*)$. Then h is not a dialgebra homomorphism: we have $z(\mathtt{F}h(*, x)) = z(h(*), h(x)) = z(h(*), h(*)) = z(\mathtt{F}h(*, *))$, while $\mathtt{B}h(f(*, x)) = \{h(*)\} \neq \emptyset = \mathtt{B}h(f(*, *))$. Intuitively, the element $*$ behaves differently from every other element of Z, but Z ought to encompass all the possible behaviours, which is a contradiction. Similarly, for any X, define the dialgebra $g(x, x) = \{x\}$, $g(x, y) = \emptyset$ if $x \neq y$. Since X is arbitrary, and no different elements are bisimilar, the cardinality of a final dialgebra is unbounded.

Final semantics is a well-established way to define behavioural equivalence of systems. In the absence of a final object in $\mathit{Dialg}(\mathtt{F}, \mathtt{B})$, we can still define behavioural equivalence by reasoning in terms of quotients of a system. Recall that a *quotient* of an object X in a category is the canonical representative of an equivalence class of epimorphisms from X, under the equivalence relation $f : X \to Y \equiv g : X \to Z$ if and only if there is an isomorphism $i : Y \to Z$ such that $i \circ f = g$. In `Set`, the quotients of an object form a set.

Definition 3. *The* bisimilarity quotient *of a dialgebra* (X, f) *is the wide pushout* (Q, q) *(if it exists) of the cone of* quotients *of* (X, f) *in* $\mathit{Dialg}(\mathtt{F}, \mathtt{B})$*. We call the diagonal* $z : (X, f) \to (Q, q)$ *the* canonical map *of* (X, f)*.*

Proposition 2. *Let* (Q, q) *be the bisimilarity quotient of* (X, f) *and* z *the canonical map. For all* $x, y \in X$*,* x *is bisimilar to* y *if and only if* $z(x) = z(y)$*. Therefore, when the bisimilarity quotient exists, bisimilarity is an equivalence relation.*

When the bisimilarity quotient exists, the canonical map can be considered the semantics of a system. Although z is canonical, it is not necessarily the unique $z : (X, f) \to (Q, q)$: bisimilarity classes may be interchangeable in dialgebras.

Example 2. Consider the dialgebra g defined in Example 1; the dialgebra has no non-trivial quotients, so it coincides with its bisimilarity quotient. However all the isomorphisms of the carrier set are dialgebra homomorphisms. The bisimilarity classes are the same, but for the identity of the sole element of each class.

For a different example, in the semantics of a symmetric process calculus such as the *pure* variant of CCS, the bisimilarity classes of an element, and of the element obtained by replacing all the input or output actions in it with the complementary ones, can be interchanged.

Clearly, when a final object exists, the epi-mono factorisation of the final morphism yields the bisimilarity quotient. A bisimilarity quotient may exist also in the absence of a final object. Here is a sufficient condition.

Proposition 3. *When* `F` *preserves wide (small) pushouts of epimorphisms, that is, colimits of an arbitrary small cone of epis, the bisimilarity quotient exists.*

The dialgebraic semantics of CCS in §5.2 is an example where the bisimilarity quotient exists, but there is no final dialgebra (by Example 1).

3 Interaction and Observation

In this section we introduce two functors $\mathtt{F}$ and $\mathtt{B}$ that can be used to give dialgebraic semantics to interactive, non-deterministic calculi.

Definition 4. *The* interaction *and* observation *functors are defined as* $\mathtt{F}(X) = X + (X \times X)$ *and* $\mathtt{B}(X) = \mathcal{P}(X)$*, respectively.*

As elements of X and $X \times X$ are syntactically disjoint, we use them to denote elements of $\mathtt{F}(X)$, avoiding labels for the coproduct. An element of $\mathtt{F}X$ is either $x \in X$, representing an experiment about a process in isolation, or $(x, y) \in X \times X$, an experiment where two processes are allowed to interact. Elements of $\mathtt{B}$ are sets of processes, that are the possible non-deterministic outcomes of an experiment. We write $x \to z$ and $(x, y) \to z$ for $z \in f(x)$ and $z \in f(x, y)$, respectively.

Proposition 4. *The functors* $\mathtt{F}$ *and* $\mathtt{B}$ *preserve monos from the empty set.* $\mathtt{F}$ *preserves wide pushouts of epis. Therefore Dialg*$(\mathtt{F}, \mathtt{B})$ *has epi-mono factorisations, and each dialgebra in the category has a bisimilarity quotient.*

Theorem 1 provides a characterisation of bisimilarity in *Dialg*$(\mathtt{F}, \mathtt{B})$.

Theorem 1. *Let* (X, f) *be an* $(\mathtt{F}, \mathtt{B})$*-dialgebra. An equivalence relation* $\mathcal{R} \subseteq X \times X$ *is the kernel of* $h : (X, f) \to (Y, g)$ *for some* h*,* Y*,* g*, if and only if, for all* $(x_1, x_2), (y_1, y_2) \in \mathcal{R}$ *and* $z_1 \in X$*, we have:* $x_1 \to z_1 \implies \exists z_2.x_2 \to z_2 \wedge (z_1, z_2) \in \mathcal{R}$*, and,* $(x_1, y_1) \to z_1 \implies \exists z_2.(x_2, y_2) \to z_2 \wedge (z_1, z_2) \in \mathcal{R}$*. As a corollary, the bisimilarity quotient of* (X, f) *is the largest such relation.*

4 Comparing Dialgebras

Categories of algebras or coalgebras of different functors may be compared by mapping one category into the other by composition with an appropriate natural transformation [6]. In §4.1 we generalise this technique to dialgebras (of which algebras and coalgebras are special cases). The problem has first been studied in [7]. Here we improve on it by adding an intermediate "container" functor $\mathtt{G}$, whose role is crucial, e.g., for §5.3. In §4.2 we discuss the limitations of this method in the setting of dialgebras, and refine the construction.

4.1 Comparing Categories of Dialgebras

Consider $\mathtt{Set}$ endofunctors $\mathtt{F}, \mathtt{B}, \mathtt{F}', \mathtt{B}'$. One can specify functor $\mathtt{K} : \mathit{Dialg}(\mathtt{F}, \mathtt{B}) \to \mathit{Dialg}(\mathtt{F}', \mathtt{B}')$ using $\mathtt{G} : \mathtt{Set} \to \mathtt{Set}$ and two natural transformations $\lambda : \mathtt{F}' \to \mathtt{GF}$ and $\mu : \mathtt{GB} \to \mathtt{B}'$, as illustrated by the diagram in Figure 2.

Notice that dialgebras come equipped with the "underlying set" or "forgetful" functor $\mathtt{U}_{\mathtt{F},\mathtt{B}} : \mathit{Dialg}(\mathtt{F}, \mathtt{B}) \to \mathtt{Set}$ defined as $\mathtt{U}_{\mathtt{F},\mathtt{B}}(X, f) = X$, $\mathtt{U}_{\mathtt{F},\mathtt{B}}(h : (X, f) \to (Y, g)) = h$; this allows us to state that $\mathtt{K}$ is *concrete*.

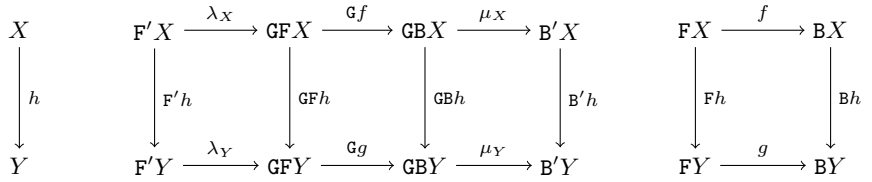


Fig. 2. Comparing categories of dialgebras using natural transformations

Theorem 2. *Two natural transformations* $\lambda : \mathtt{F}' \to \mathtt{GF}$ *and* $\mu : \mathtt{GB} \to \mathtt{B}'$ *determine a functor* $\mathtt{K} : \mathit{Dialg}(\mathtt{F},\mathtt{B}) \to \mathit{Dialg}(\mathtt{F}',\mathtt{B}')$ *as* $\mathtt{K}(X,f) = (X, \mu_X \circ \mathtt{G}f \circ \lambda_X)$, $\mathtt{K}(h : (X,f) \to (Y,g)) = h$. K *is concrete, that is:* $\mathtt{U}_{\mathtt{F}',\mathtt{B}'} \circ \mathtt{K} = \mathtt{U}_{\mathtt{F},\mathtt{B}}$.

As a consequence of K being concrete, we have the following corollary.

Corollary 1. *If* $x, y \in X$ *are bisimilar in* (X,f) *then they are so in* $\mathtt{K}(X,f)$.

4.2 Comparing Dialgebras

The framework of §4.1 is more restrictive than necessary. Observe that λ and μ have to be defined for each set X. But when comparing say, two different semantics of a language, we are only interested in the two dialgebras, and in the objects that may be reached from them by an epimorphism. Considering this subclass of objects, in the definition of λ and μ, we could use specific features of a given dialgebra (e.g. exploit the fact that the underlying set X is the carrier of an algebra, or refer to specific elements of X). In this light, we now restrict our attention to natural transformations between functors whose codomain is Set, but whose domain is not. The framework appears complicated at first; however, Theorem 3 allows us to specify such natural transformations by single functions, called *bisimulation invariants*, that serve as an "adaptation layer" between dialgebras of different type. Although proofs in this section require quite a bit of categorical reasoning, defining a bisimulation invariant is not a difficult task by itself and encapsulates the complexity of the framework in a simple definition. We shall support this claim in §5.3, which proves equivalence of the dialgebraic and coalgebraic semantics of CCS.

Let E be the subcategory of epimorphisms of $\mathit{Dialg}(\mathtt{F},\mathtt{B})$. Given a dialgebra (X,f), consider the coslice category $(X,f)/\mathtt{E}$. Its objects are arrows in E whose domain is (X,f). Its arrows are commuting morphisms of E. Our framework is parametrised by a full subcategory of $(X,f)/\mathtt{E}$ having the following properties.

Definition 5. *Let* $\mathtt{R}_{(X,f)}$ *be a full subcategory of* $(X,f)/\mathtt{E}$*, such that: for each object* h *of* $(X,f)/\mathtt{E}$ *there is at least one object* h' *of* $\mathtt{R}_{(X,f)}$ *with a commuting arrow* $k : h \to h'$*; the identity* $id_{(X,f)}$ *is an object.*

By the first condition, $\mathtt{R}_{(X,f)}$ contains enough epimorphisms to characterise bisimilarity: whenever x and y are identified by a morphism h, there is h' in the

category that identifies them. The second condition embeds the object (X, f) into $\mathtt{R}_{(X,f)}$. The purpose of $\mathtt{R}_{(X,f)}$ is to serve as a domain for functors into $\mathtt{Set}$, so that natural transformations between them may be more specific, while preserving bisimilarity of the object (X, f). A natural transformation in this setting may make explicit reference to (X, f). Notice that the maps composing such a natural transformation depend on dialgebra morphisms, not objects, by definition of $\mathtt{R}_{(X,f)}$. Using a *subcategory* of $(X, f)/\mathtt{E}$ further relaxes proof obligations; e.g., in Proposition 5 we only consider morphisms whose kernels are congruences with respect to the parallel operator. The following definition casts $\mathtt{Set}$ endofunctors into functors from $\mathtt{R}_{(X,f)}$ to $\mathtt{Set}$.

Definition 6. *For each* $\mathtt{F} : \mathtt{Set} \to \mathtt{Set}$*, define its* lifting $\bar{\mathtt{F}} : \mathtt{R}_{(X,f)} \to \mathtt{Set}$ *as* $\bar{\mathtt{F}} = \mathtt{F} \circ \mathtt{U}_{\mathtt{F},\mathtt{B}} \circ \mathtt{cod}$*, where* $\mathtt{cod} : (X, f)/\mathtt{E} \to \mathtt{E}$ *is the codomain functor, mapping objects (arrows in Dialg*$(\mathtt{F}, \mathtt{B})$*) to their codomains, and arrows to themselves.*

Spelling out the definition, $\bar{F}$ acts on objects as $\bar{\mathtt{F}}(p : (X, f) \to (Y, g)) = F(Y)$ and on arrows as $\bar{F}(k) = k$. Next, we prove that natural transformations indexed by $\mathtt{R}_{(X,f)}$ may be specified by single functions, obeying to a condition that we call *bisimulation invariance*. Notice that, since $\mathtt{R}_{(X,f)}$ is a full subcategory containing the identity of a coslice category, each arrow $h : (X, f) \to (Y, g)$ can be regarded as both an object of $\mathtt{R}_{(X,f)}$ and an arrow in the same category from $id_{(X,f)}$ to h itself. In the following, we refer to the arrow as $\hat{h}$ to avoid confusion.

Definition 7. *Given two functors* $\mathtt{F}, \mathtt{G} : \mathtt{R}_{(X,f)} \to \mathtt{Set}$*, consider a function* $k : \mathtt{F}(id_{(X,f)}) \to \mathtt{G}(id_{(X,f)})$*. Call* k bisimulation invariant with respect to (X, f) and $\mathtt{R}_{(X,f)}$ from $\mathtt{F}$ to $\mathtt{G}$ *if and only if, for all* $x_1, x_2 \in X$*, and for each arrow* $\hat{h} : id_{(X,f)} \to h$ *in* $\mathtt{R}_{(X,f)}$*, we have* $\mathtt{F}\hat{h}(x_1) = \mathtt{F}\hat{h}(x_2) \implies \mathtt{G}\hat{h}(k(x_1)) = \mathtt{G}\hat{h}(k(x_2))$.

In the following we call k simply *invariant* when (X, f), $\mathtt{R}_{(X,f)}$, $\mathtt{F}$ and $\mathtt{G}$ are clear from the context. Such a property of a function may seem difficult to prove. However, it is actually easier to show invariance of a given function with respect to the lifted versions of two functors, than naturality of a transformation in $\mathtt{Set}$ between the same functors. One needs to prove commutativity just for a given class of morphisms (those in $\mathtt{R}_{(X,f)}$), which are also guaranteed to preserve and reflect bisimilarity. The fundamental property of invariants is Theorem 3, that depends on $\mathtt{F}$ preserving epis. This holds for the lifting $\bar{\mathtt{F}}$ of a $\mathtt{Set}$ endofunctor, as all $\mathtt{Set}$ endofunctors preserve epis, and so do $\mathtt{cod}$ and $\mathtt{U}$ used in Definition 6.

Theorem 3. *Consider two functors* $\mathtt{F}, \mathtt{G} : \mathtt{R}_{(X,f)} \to \mathtt{Set}$*, with* $\mathtt{F}$ *preserving epis. There is a one-to-one correspondence between natural transformations* $\delta : \mathtt{F} \to \mathtt{G}$ *and invariants from* $\mathtt{F}$ *to* $\mathtt{G}$*. Each natural transformation* δ *is uniquely determined by* $\delta_{id(X,f)}$*, which is invariant; conversely, for each invariant* k *there is a unique natural transformation* δ *such that* $\delta_{id(X,f)} = k$.

We can restate Theorem 2 in terms of natural transformations λ, μ between functors from $\mathtt{R}_{(X,f)}$ to $\mathtt{Set}$; this is described by the diagram in Figure 3.

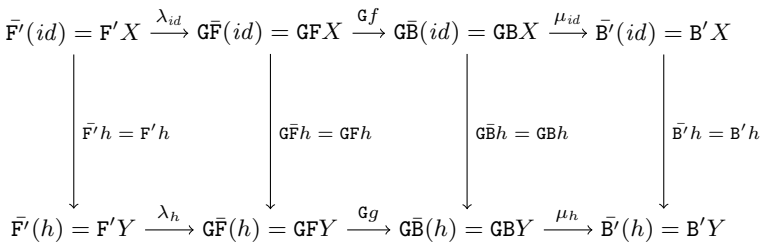


Fig. 3. Comparing dialgebras using functors from $\mathtt{R}_{(X,f)}$ to $\mathtt{Set}$. Here id is $id_{(X,f)}$.

Theorem 4. *Given an* $(\mathtt{F},\mathtt{B})$*-dialgebra* (X,f)*, a category* $\mathtt{R}_{(X,f)}$ *as in Definition 7, a functor* $\mathtt{G}$*, and two invariants* λ *from* $\bar{\mathtt{F}}'$ *to* $\mathtt{G}\bar{\mathtt{F}}$*,* μ *from* $\mathtt{G}\bar{\mathtt{B}}$ *to* $\bar{\mathtt{B}}'$*, consider the* $(\mathtt{F}',\mathtt{B}')$*-dialgebra* $(X, f^{\lambda,\mu})$ *where* $f^{\lambda,\mu} = \mu \circ \mathtt{G}f \circ \lambda$*. Whenever two elements are bisimilar in* (X,f)*, then they are bisimilar in* $(X, f^{\lambda,\mu})$*.*

5 Examples

5.1 The Coalgebraic Semantics of CCS

The *Calculus of Communicating Systems* (CCS) is a simple process calculus, formalising a fundamental aspect of computation: *communication* between parallel processes. In the *pure* variant only *synchronisation* is considered, that is, the exchanged data is not taken into account. We briefly recall the LTS (thus, coalgebraic) semantics of CCS here. The interested reader may refer to [13] for more details. The syntax is described by the grammar:

$$P ::= \sum_{i\in I} \alpha_i.P_i \mid P_1 \parallel P_2 \mid (\nu a)P \qquad\qquad \alpha ::= \tau \mid a \mid \bar{a}$$

where I is a finite set, and a ranges over a countable set of channels $\mathcal{C}$. Elements of P are *processes*, or *agents*. Elements of α are *atomic actions*, or *prefixes*, or *guards*. CCS features operators for denoting: *parallel composition* $(P_1 \parallel P_2)$; *restriction* of a channel x which becomes private to P $((\nu a)P)$; non-deterministic *guarded choice* among a finite set of action-prefixed processes $(\sum_{i\in I} \alpha_i.P_i)$, usually written as $a_1.P_1 + \ldots + a_n.P_n$. Special cases of the choice construct are the empty process $\emptyset$ which is the sum of zero processes, and the action prefix $\alpha.P$, which is the sum of one process. Notice that choice is commutative by construction. The actions α are: the *internal step* (τ); the act of *receiving* a signal on channel a (the action a); *sending* a signal on a channel $(\bar{a})$. We omit recursion for simplicity; including it does not change the presented results. Channels are also called *names*. *Free* names $fn(-)$ are defined by induction, as usual, for processes and labels. The only binding construct is $(\nu a)P$, in which name a is *bound*. In the following, let X be the set of CCS processes.

Definition 8. Structural congruence *is the minimal congruence* $\equiv \subseteq X \times X$ *including* α*-conversion of* a *in* $(\nu a)P$*; commutative monoid axioms for the parallel operator with respect to* 0*; the equations* $(\nu a)(P \parallel Q) \equiv ((\nu a)P) \parallel Q$*,* $(\nu a)\emptyset \equiv \emptyset$*,* $(\nu a)(\nu b)P \equiv (\nu b)(\nu a)P$ *for all* P*,* Q*,* $a \notin fn(Q)$ *and* b*.*

The labelled transition system for CCS is presented in Figure 4. The set L of labels is just the set of prefixes α. We write $x \xrightarrow{\alpha} y$ as a shorthand for $(\alpha, y) \in g(x)$. An LTS with labels from L can be represented as a pair (X, f) where f is a function[2] from X to $\mathcal{P}(L \times X)$. Let (X, g) be the LTS for CCS.

$$\alpha.P + Q \xrightarrow{\alpha} P \,(pre) \quad \frac{a \notin fn(\alpha) \quad P \xrightarrow{\alpha} P'}{(\nu a)P \xrightarrow{\alpha} (\nu a)P'}\,(res) \quad \frac{P \xrightarrow{\alpha} P'}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q}\,(par)$$

$$\frac{P \xrightarrow{\bar{c}} P' \quad Q \xrightarrow{c} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}\,(syn) \quad \frac{P \equiv Q \quad P' \equiv Q' \quad P \xrightarrow{\alpha} P'}{Q \xrightarrow{\alpha} Q'}\,(str)$$

Fig. 4. The LTS describing the operational semantics of CCS

Definition 9. *CCS bisimilarity is the greatest symmetric relation* $\sim_g \subseteq X \times X$ *such that, if* $x \sim_g y$ *and* $x \xrightarrow{\alpha} x'$*, there is* y' *such that* $y \xrightarrow{\alpha} y'$ *and* $x' \sim_g y'$*.*

Notice that we are defining bisimilarity in one specific system (e.g. CCS), not between states of different systems. This corresponds to a standard categorical notion once recognised that LTSs are $\mathcal{P}(L \times -)$-coalgebras.

Definition 10. *Given a functor* $\mathtt{T} : \mathtt{Set} \to \mathtt{Set}$*, a* $\mathtt{T}$*-coalgebra is a pair* $(X, f : X \to \mathtt{T}X)$ *where* X *is a set. A homomorphism of coalgebras from* (X, f) *to* (Y, g) *is a function* $h : X \to Y$ *such that* $g \circ h = \mathtt{T}h \circ f$*.* $\mathtt{T}$*-coalgebras and their morphisms form the category Coalg*$(\mathtt{T})$*.*

Definition 11. *Given a* $\mathtt{T}$*-coalgebra* (X, f)*,* coalgebraic bisimilarity $\sim_f \subseteq X \times X$ *is defined by* $x \sim_f y \iff \exists (Y, g).\exists h : (X, f) \to (Y, g).h(x) = h(y)$*.*

It is well-known that under suitable conditions on $\mathtt{T}$ (that the functor for LTSs respects) bisimilarity as in Definition 11 coincides with other coalgebraic notions (see e.g. [14]). We do not discuss the details, but we note that for LTSs and the one for CCS in particular, the relations from Definition 9 and 11 coincide.

5.2 A Dialgebraic Semantics of CCS

In §5.1, we have seen the coalgebraic semantics of CCS. Now we describe it as an $(\mathtt{F}, \mathtt{B})$-dialgebra for the functors of Definition 4.

[2] Here $\mathcal{P}(X)$ is the *(co-variant) power set* of X. In coalgebras it is typical to assume a cardinality bound on the size of the subsets, as the functor $\mathcal{P}$ does not have a final coalgebra. Indeed, in all the systems we define, $\mathcal{P}$ can be replaced by a bounded variant, as all our transition sets are finite or countable. Since we do not use the final coalgebra in this paper, the distinction is immaterial here.

Definition 12. *Let X be the set of CCS processes. The dialgebra (X, f) for CCS is the least function obeying to the rules in Figure 5.*

$$\tau.P + Q \to P\ (tau) \qquad \frac{P \to P'}{(\nu a)P \to (\nu a)P'}\ (res) \qquad \frac{P \to P'}{P \parallel Q \to P' \parallel Q}\ (par_1) \qquad \frac{(P,Q) \to R}{P \parallel Q \to R}\ (int)$$

$$\frac{(P,Q) \to R \quad a \notin fn(Q)}{((\nu a)P, Q) \to (\nu a)R}\ (hid) \qquad \frac{(P,Q) \to R}{(P \parallel S, Q) \to S \parallel R}\ (par_2) \qquad (\bar{a}.P + S, a.Q + T) \to P \parallel Q\ (syn)$$

$$\frac{(P,Q) \to R}{(Q,P) \to R}\ (sym) \qquad \frac{P \to R \quad P \equiv Q, R \equiv S}{Q \to S}\ (str_1) \qquad \frac{(P,Q) \to R \quad P \equiv S, Q \equiv T, R \equiv U}{(S,T) \to U}\ (str_2)$$

Fig. 5. The dialgebra for CCS

We briefly comment on the rules. The first group deals with processes in isolation: rule (tau) permits internal computation actions to be executed; Rule (res) allows a process in the scope of a restriction to progress; Rule (par_1) allows one component in a parallel composition to progress independently from the others; Rule (int) allows any possible interaction between two processes to also happen between internal components of a process in isolation.

The second group of rules defines the semantics of interaction. Rule (hid) permits interaction between a process P in the scope of a restriction, and any other process Q, provided that a is not known by Q. Recall that the restricted name a can always be α-converted to one which is fresh in Q. Rule (par_2) allows parallel components of a process P to interact with Q independently from each other. Rule (syn) implements synchronisation between two processes.

Rules (sym), (str_1), (str_2) simplify the definition; alternatively, one can add variants of the other rules taking into account the effects of these three schemes.

5.3 Comparing the Coalgebraic and Dialgebraic Semantics of CCS

In this section, we use Theorem 4 to compare the semantics for CCS from Definition 12 to bisimilarity in the well-known labelled transition system.

First, we attempt to give some intuition on the constructions of §4. Consider defining a coalgebra (X, f') for the functor of §5.1 out of the dialgebra for CCS of §5.2. We can employ "witness processes" such as $a.0$ in experiments such as $(x, a.0)$. For each $x' \in f(x, a.0)$, we let $f'(x)$ contain the labelled transition $(\bar{a}, x')$. The idea sounds promising, but in the process we need to refer to specific elements of X, namely the witness processes, thus to the specific set X of elements. Natural transformations are not allowed to depend upon an object in this way; therefore, we need the the theory of §4.

The dialgebra of §5.2 describes processes as they interact, with no explicit notion of side effect. The coalgebra of §5.1 describes processes in isolation, and their side effects. It is not difficult to imagine how the two kinds of semantics

can be compared. In one direction, starting from the coalgebra, we may define a $(\mathtt{F},\mathtt{B})$-dialgebra on the same carrier; for processes in isolation, we run one step of the LTS, and then turn all the τ transitions into observed results; for interaction between pairs of elements, we run one step of the LTS on each element, and let interaction happen whenever an input (or an output) is matched by the complementing action. In the other direction, starting from the dialgebra, we may define a $\mathtt{T}$-coalgebra, by letting τ transitions correspond to the observations that are made on a process in isolation, and by running experiments in which we let a process and a "witness process" such as $a.\emptyset$ interact. The resulting transitions are labelled with a corresponding action, e.g., $\bar{a}$ in our case.

In the rest of the section we closely implement the above plan. From now on, we let (X, f), (X, g), and the functors $\mathtt{F},\mathtt{B}$ be the dialgebra and the coalgebra for CCS, and the functors from §3; we let $\mathtt{T}(X) = \mathcal{P}(L \times X)$. Below, we define three invariants λ, μ, δ, that we shall use to instantiate Theorem 4 twice. In one direction, we will define the coalgebra $\mu \circ \mathtt{T}f \circ \lambda$ out of the dialgebra f. In the other direction, we will get the dialgebra $\delta \circ \mathtt{F}g \circ id_{\mathtt{F}X}$ from g.

Definition 13. *We define* $\delta : \mathtt{FT}X \to \mathtt{B}X$, $\lambda : X \to \mathtt{TF}X$, $\mu : \mathtt{TB}X \to \mathtt{T}X$ *as:*

$$\delta(e) = \begin{cases} \{x \mid (\tau, x) \in p\} & \text{if } e = p \in \mathtt{T}X \\ \{x \parallel y \mid \exists a \in \mathcal{C}.\, ((a,x) \in p_1 \wedge (\bar{a}, y) \in p_2) \vee & \text{if } e = (p_1, p_2) \in \mathtt{T}X \times \mathtt{T}X \\ \qquad \vee\, ((\bar{a}, x) \in p_1 \wedge (a, y) \in p_2)\} & \end{cases}$$

$$\lambda(x) = \{(\tau, x)\} \cup \{(a, (x, \bar{a}.\emptyset)) \mid a \in \mathcal{C}\} \cup \{(\bar{a}, (x, a.\emptyset)) \mid a \in \mathcal{C}\}$$

$$\mu(q) = \{(l, x) \mid \exists q'.(l, q') \in q \wedge x \in q'\}$$

Notice how the definition of δ uses the fact that X is also the carrier of the initial algebra, therefore the parallel composition $x \parallel y$ is defined. An appropriate choice of $\mathtt{R}_{(X,g)}$ makes δ an invariant. Also, the definition of λ uses specific elements of X, such as $\bar{a}.\emptyset$. On the other hand, μ is independent of X and extends to a natural transformation from $\mathtt{TB}$ to $\mathtt{T}$.

Proposition 5. *Let E be the subcategory of Dialg$(\mathtt{F},\mathtt{B})$ of epis whose domain is (X, f), and E' the subcategory of Dialg$(\mathtt{Id},\mathtt{T})$ of epis whose domain is (X, g). Let $\mathtt{R}_{(X,f)}$ be the coslice $(X, f)/E$, and $\mathtt{R}_{(X,g)}$ be the full subcategory of $(X, g)/E'$ whose objects h commute with the parallel operator, that is, $h(x) = h(x') \wedge h(y) = h(y') \implies h(x \parallel y) = h(x' \parallel y')$. Then:*

- $\mathtt{R}_{(X,g)}$ *obeys to Definition 5;*
- $id_{\mathtt{F}X}$ *is invariant for* (X, g) *and* $\mathtt{R}_{(X,g)}$ *from* $\bar{\mathtt{F}}$ *to* $\mathtt{F}\,\bar{\mathtt{Id}} = \bar{\mathtt{F}}$*;*
- δ *is invariant for* (X, g) *and* $\mathtt{R}_{(X,g)}$ *from* $\mathtt{F}\bar{\mathtt{T}}$ *to* $\mathtt{B}$*;*
- λ *is invariant for* (X, f) *and* $\mathtt{R}_{(X,f)}$ *from* $\bar{\mathtt{Id}}$ *to* $\mathtt{T}\bar{\mathtt{F}}$*;*
- μ *is invariant for* (X, f) *and* $\mathtt{R}_{(X,f)}$ *from* $\mathtt{T}\bar{\mathtt{B}}$ *to* $\bar{\mathtt{T}}$*.*

In Proposition 5, $\mathtt{R}_{(X,g)}$ contains only homomorphisms that commute with the parallel operator, strengthening the hypothesis for invariance, which facilitates

the proof. We can now use Theorem 4, twice. Let $\mathtt{G} = \mathtt{F}$; we obtain the $(\mathtt{F}, \mathtt{B})$-dialgebra $(X, g^{id_{\mathtt{F}X},\delta})$ where $f^{id_{\mathtt{F}X},\delta} = \delta \circ \mathtt{F}g$. Similarly, let $\mathtt{G} = \mathtt{T}$; then we derive the $(\mathtt{Id}, \mathtt{T})$-dialgebra, that is, $\mathtt{T}$-coalgebra, $(X, f^{\lambda,\mu})$ with $f^{\lambda,\mu} = \mu \circ \mathtt{T}f \circ \lambda$.

So far, we have mapped the dialgebra of CCS into a coalgebra, and the coalgebra into an $(\mathtt{F}, \mathtt{B})$-dialgebra. However, no link is established between f and $g^{id,\delta}$, or g and $f^{\lambda,\mu}$. We conclude the paper by proving coincidence of the two semantics. For this, we need the following lemma.

Lemma 1. *For all channels a and elements x, y, z, the following holds: $x \xrightarrow{\tau} y \iff x \to y$; $x \xrightarrow{a} y \iff (x, \bar{a}.\emptyset) \to y$; $x \xrightarrow{\bar{a}} y \iff (x, a.\emptyset) \to y$; $(x, y) \to z \iff \exists b, x', y'. z \equiv x' \parallel y' \wedge ((x \xrightarrow{b} x' \wedge y \xrightarrow{\bar{b}} y') \vee (x \xrightarrow{\bar{b}} x' \wedge y \xrightarrow{b} y'))$.*

Proposition 6. *Let (X, f) and (Y, g) be the dialgebra and the coalgebra for CCS. We have that $f = g^{id_{\mathtt{F}X},\delta}$ and $g = f^{\lambda,\mu}$. Therefore, bisimilarity in (X, f) and in (X, g) is the same.*

5.4 The π-Calculus

The π-calculus [15] is a very well known extension of CCS. The calculus takes network mobility into account by the means of fresh name generation and communication. Bisimilarity in the π-calculus is non-standard, since it requires side conditions on freshness of names that do not permit one to compare labels just syntactically. A coalgebraic semantics is possible by switching from the category `Set` to presheaves (see e.g. [1]). In this section we provide a dialgebraic semantics to the calculus. Remarkably, the dialgebra we define lives in `Set`; the difference from the semantics of CCS is just to add data passing in the rule for synchronisation. We give a very brief summary of the calculus here. The reader may consult e.g., [16] for further information. The π-calculus features data passing, and fresh name creation. Channels and data coincide, giving to the calculus its expressive power. The syntax is as follows.

$$P ::= \sum_{i \in I} \alpha_i.P_i \mid P_1 \parallel P_2 \mid (\nu a)P \qquad\qquad \alpha ::= \tau \mid a(x) \mid \bar{a}x$$

Again, we do not introduce recursion, as it does not add to the presentation and complicates proofs. In the syntax, a, x, y range over a countable set of *channel names*. The prefix $a(x)$ reads x from channel a. Therefore, x is bound in $a(x).P$. The prefix $\bar{a}x$ sends x on channel a. The other constructs have the same informal meaning as in the CCS, and share the same syntax. We adopt the *early* semantics of the calculus. For space reasons, we omit the definition of the corresponding transition system, which is widely available (see e.g. [16], Definition 1.3.2). We just mention that the transitions may have four kinds of labels: τ, $\bar{a}b$, ab, $\bar{a}(x)$, corresponding to silent actions, output of b on channel a, input of b on a, and *bound output*, where b is a fresh name. Synchronization with a process doing bound output may only take place when b is fresh in the receiving process, which is obtained by α-conversion of b.

The close syntactic resemblance between CCS and the π-calculus is reflected in the dialgebraic semantics we propose, as we only need to change one rule to switch from one semantics to the other. The formal definition uses structural congruence, which is the same as Definition 8, with the addition of α-conversion of variable x for processes under the scope of an input $a(x)$.

Definition 14. *Let X be the set of π-calculus terms. Define the $(\mathtt{F},\mathtt{B})$-dialgebra (X, f) importing the rules of Definition 12, where Rule (syn) is replaced by*

$$(\bar{a}b.P + S, a(y).Q + T) \to P \parallel Q[^{b}/_{y}] \quad (com)$$

Rule (*com*) models data passing in the usual way. Combined with Rule (*hid*), it implicitly handles *scope extrusion*, which is one of the most difficult bits of the π-calculus semantics. For example, for x fresh in Q, we have $((\nu x)\bar{a}x.P, a(y).Q) \to (\nu x)(P \parallel Q[^{x}/_{y}])$, by first applying (*hid*) and then (*com*). Such a simple treatment of scope extrusion is inherited from the reactive system for the π-calculus that we are mimicking (see [16], Definition 1.2.12). The dialgebraic definition adds to it a non-trivial notion of bisimilarity, that we ought to relate to the standard definition. A direct comparison, as in §5.3, is not obvious, as the coalgebraic semantics lives in a presheaf category, whereas the dialgebraic semantics is defined in `Set`. However, we are able to reuse well known results for the π-calculus to obtain a characterisation theorem. Dialgebraic semantics is easier to compare to *strong barbed equivalence*, defined below, than to bisimilarity. We use the *observability predicate* $P \downarrow_{\mu}$, for μ in the form a or $\bar{a}$, that holds whenever P can perform a communication action $a(x)$ or $\bar{a}x$, respectively. In turn, strong barbed equivalence is defined in terms of *strong barbed bisimilarity*.

Definition 15. Strong barbed bisimilarity *is the largest symmetric relation $\dot{\sim}$ such that whenever $P \dot{\sim} Q$, $P \downarrow_{\mu} \implies Q \downarrow_{\mu}$, and $P \xrightarrow{\tau} P' \implies \exists Q'.Q \xrightarrow{\tau} Q'$ with $P' \dot{\sim} Q'$. We say that processes P and Q are* strong barbed equivalent*, written $P \simeq Q$, if for all R, $P \parallel R \dot{\sim} Q \parallel R$.*

Theorem 5. *Dialgebraic bisimilarity coincides with strong barbed equivalence.*

Finally, Theorem 2.2.9 in [16] proves that strong barbed equivalence coincides with strong early bisimilarity[3]. Thus, as a corollary, we get that dialgebraic bisimilarity coincides with early bisimilarity.

6 Conclusions and Future Work

The most important difference between coalgebras and dialgebras is that there is no final dialgebra, therefore no universal model. This forces one to reason in terms of quotients. The locality which is intrinsic to the definition of a dialgebra deserves in our opinion a more thorough investigation in various directions.

[3] Therein, it is shown that the *matching* prefix of the π-calculus is not required for this result to hold, thus we omit it for simplicity.

A fundamental problem is to spell out an inductive definition principle, in order to obtain simpler definitions, and compositionality. A conjecture on how to generalise the use of distributive laws [17] from bialgebras to dialgebras has been formulated in [18], and will be developed in future work.

Logical aspects should also be considered. The interplay between adequate logics for dialgebras, and equational logic on terms, may lead to new insights on algebraic and coalgebraic specifications.

Another matter is the implementation and verification of dialgebras. Coalgebras have an associated partition refinement procedure that computes the bisimilarity quotient of a system, by the means of iteration along the terminal sequence of the functor `T`. A generalisation of this procedure to dialgebras appears in [18], and will be explained and enhanced in future work.

Finally, an open question is the definition of a proof principle for dialgebras, generalising induction and coinduction.

References

1. Fiore, M.P., Turi, D.: Semantics of name and value passing. In: 12th Annual Symposium on Logic in Computer Science (LICS), pp. 93–104. IEEE (2001)
2. Trnková, V., Goralcík, P.: On products in generalized algebraic categories. Commentationes Mathematicae Universitatis Carolinae 10(1), 49–89 (1969)
3. Adámek, J.: Limits and colimits in generalized algebraic categories. Czechoslovak Mathematical Journal 26(1), 55–64 (1976)
4. Hagino, T.: A Categorical Programming Language. PhD thesis, University of Edinburgh (1987)
5. Poll, E., Zwanenburg, J.: From algebras and coalgebras to dialgebras. Electronic Notes in Theoretical Computer Science 44(1), 289–307 (2001)
6. Rutten, J.J.M.M.: Universal coalgebra: a theory of systems. Theoretical Computer Science 249(1), 3–80 (2000)
7. Voutsadakis, G.: Universal dialgebra: unifying algebra and coalgebra. Far East Journal of Mathematical Sciences 44(1), 1–53 (2010)
8. Madeira, A., Martins, M.A., Barbosa, L.: Models as arrows: the role of dialgebras. In: 7th Conference on Computability in Europe, Sofia, Bulgaria, pp. 144–153 (2011)
9. Ciancia, V.: Interaction and observation, categorically. In: 4th Interaction and Concurrency Experience. EPTCS, vol. 59, pp. 25–36 (2011)
10. Leifer, J.J., Milner, R.: Deriving bisimulation congruences for reactive systems. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 243–258. Springer, Heidelberg (2000)
11. Sewell, P.: From rewrite rules to bisimulation congruences. Theoretical Computer Science 274, 183–230 (2002)
12. Sassone, V., Sobocinski, P.: Deriving bisimulation congruences using 2-categories. Nordic Journal of Computing 10(2), 163–186 (2003)
13. Milner, R. (ed.): A Calculus of Communication Systems. LNCS, vol. 92. Springer, Heidelberg (1980)
14. Staton, S.: Relating coalgebraic notions of bisimulation. Logical Methods in Computer Science 7(1) (2011)

15. Milner, R., Parrow, J., Walker, D.: A Calculus of Mobile Processes, Part I. Information and Computation 100(1), 1–40 (1992)
16. Sangiorgi, D., Walker, D.: The π-Calculus: A Theory of Mobile Processes. Cambridge University Press, New York (2001)
17. Turi, D., Plotkin, G.: Towards a mathematical operational semantics. In: 12th Annual Symposium on Logic in Computer Science (LICS), pp. 280–291. IEEE (1997)
18. Blok, A.: Interaction, observation and denotation: A study of dialgebras for program semantics. Master's thesis, University of Amsterdam (2012)

Homomorphisms of Coalgebras from Predicate Liftings

Sebastian Enqvist[1,2]

[1] Department of Philosophy, Lund University
[2] ILLC, University of Amsterdam

Abstract. This paper assumes the concept of a predicate lifting from coalgebraic modal logic, and associates with every set Λ of predicate liftings for a set functor T a category $\mathbb{C}_T^\Lambda$ of T-coalgebras and socalled *Λ-homomorphisms*. From this construction, some natural constructions on models such as *products of models* and *submodels* can be defined. A relationship with simulations of coalgebras arising from lax extensions is established, and the main technical result gives a condition under which the category $\mathbb{C}_T^\Lambda$ is both complete and cocomplete.

1 Technical Preliminaries

1.1 Coalgebras

The basic definitions of coalgebras and coalgebra morphisms are as follows:

Definition 1. *Given an endofunctor T on* **Set** *(the category of sets and functions), a T-coalgebra is defined to be a pair (X, f) where X is a set and f is a mapping $X \to TX$. The category $\mathbb{C}_T$ of T-coalgebras is then constructed by letting a morphism $h : \mathfrak{A} \to \mathfrak{B}$, where $\mathfrak{A} = (X, f)$ and $\mathfrak{B} = (Y, g)$, be defined as a mapping $X \to Y$ such that $g \circ h = Th \circ f$.*

Here, we shall be working with two simple examples that are familiar from modal logic: Kripke models and neighborhood models (for simplicity, both with just a single modality and without propositional variables).

Definition 2. *A Kripke model is a pair (X, R) where X is a set and R is a binary relation over X, called an accessibility relation.*

Kripke models can be naturally represented as coalgebras, as follows:

Definition 3. *The covariant powerset functor $\mathcal{P}$ takes any set X to its powerset, and any function $f : X \to Y$ to the function $\mathcal{P}f : \mathcal{P}X \to \mathcal{P}Y$ defined by $\mathcal{P}f(Z) = f[Z] = \{v \in Y \mid v = f(u)$ for some $u \in Z\}$.*

A coalgebra (X, f) for $\mathcal{P}$ then naturally corresponds to a Kripke model, if we think of the members of $f(u)$ for a given $u \in X$ as the successors of u under the accessibility relation. More formally, we introduce the following notation:

R. Heckel and S. Milius (Eds.): CALCO 2013, LNCS 8089, pp. 126–140, 2013.

Definition 4. *Let (X, f) be a $\mathcal{P}$-coalgebra. Then the corresponding accessibility relation over X is denoted by R_f, i.e. we have uR_fv iff $v \in f(u)$.*

Morphisms of $\mathcal{P}$-coalgebras also have a natural interpretation, shown by the following well known proposition:

Proposition 1. *Given $\mathcal{P}$-coalgebras $\mathfrak{A} = (X, f)$ and $\mathfrak{B} = (Y, g)$, a mapping $h : X \to Y$ is a coalgebra morphism iff it is a bounded morphism from (X, R_f) to (Y, R_g), i.e. we have*

Forth *uR_fv implies $h(u)R_gh(v)$, and*
Back *if $h(u)R_gw$ then there is some v with uR_fv and $h(v) = w$*

Bounded morphisms are important in model theory of modal logic, and they are the functional version of *bisimulations*. If we drop the "Back" condition from the definition of a bounded morphism, we get the socalled *relation homomorphisms*, which are the functional equivalents of *simulations*. That is, a relation homomorphism from (X, R) to (X', R') is a mapping $h : X \to Y$ such that uRv implies $h(u)R'h(v)$.

We turn now to our second example, which generalizes Kripke models:

Definition 5. *A neighborhood model is a pair (X, f) where X is a set and f is a function that assigns, to each member u of X, a family of subsets of X, called the neighborhoods of u.*

The representation of neighborhood models as coalgebras is even more direct - a neighborhood model simply is a coalgebra. However, rather than working with the covariant powerset functor, we now use the *contravariant* powerset functor $\breve{\mathcal{P}}$ defined as follows:

Definition 6. *The contravariant powerset functor $\breve{\mathcal{P}}$ takes any set X to its powerset, and any function $f : X \to Y$ to the function $\breve{\mathcal{P}}f : \breve{\mathcal{P}}Y \to \breve{\mathcal{P}}X$ defined by $\breve{\mathcal{P}}f(Z) = f^{-1}[Z] = \{u \in X \mid f(u) \in Z\}$.*

A neighborhood model can then be thought of as a coalgebra for the *neighborhood functor* $\mathcal{N}$ defined as $\breve{\mathcal{P}} \circ \breve{\mathcal{P}}$.

Proposition 2. *Given $\mathcal{N}$-coalgebras (X, f) and (Y, g), a mapping $h : X \to Y$ is a coalgebra morphism iff, for each $u \in X$, we have*

$$g(h(u)) = \{Z \subseteq Y \mid h^{-1}[Z] \in f(u)\} \tag{1}$$

There is a natural way to relax this condition as well: say that a function $h : X \to Y$ is a *continuous mapping* from (X, f) to (Y, g) if, for all $u \in X$, $Z \in g(h(u))$ implies that $h^{-1}[Z]$ in $f(u)$. The term "continous" seems appropriate here: the inverse image of a neighborhood of $h(u)$ is a neighborhood of u. Indeed, later on we shall consider the special case of *topological coalgebras*, for which the continuous mappings are exactly continuous mappings of topological spaces.

A variant of neighborhood models that sometimes appear in the literature are coalgebras for the *monotone* neighborhood functor:

Definition 7. *The monotone neighborhood functor $\mathcal{M}$ takes a set X to the set of families $N \in \mathcal{N}X$ such that $A \subseteq B$ and $A \in N$ implies $B \in N$. For a mapping $h : X \to Y$ we define $\mathcal{M}h$ as the mapping $(\mathcal{N}h) \restriction \mathcal{M}X$, with the codomain changed to $\mathcal{M}Y$.*

The main examples in this paper will be based on coalgebras for $\mathcal{P}$, $\mathcal{N}$ and $\mathcal{M}$. "Continuous mappings" for $\mathcal{M}$-coalgebras are defined in the same way as for $\mathcal{N}$-coalgebras.

1.2 Predicate Liftings

Modal logics have been used as languages for coalgebras since the seminal paper by L. Moss [5]. A more recent approach uses the auxiliary notion of a *predicate lifting* [7]:

Definition 8. *Let T be any endofunctor on* **Set**. *A predicate lifting for T is a natural transformation $\lambda : \breve{\mathcal{P}} \to \breve{\mathcal{P}} \circ T$. A predicate lifting λ is said to be monotone if $Z \subseteq Z'$ implies $\lambda_X(Z) \subseteq \lambda_X(Z')$ for all X, Z, Z'.*

Given a set of predicate liftings Λ, we can define a corresponding modal language $\mathcal{ML}(\Lambda)$ as follows:

$$\mathcal{ML}(\Lambda) := \bot \mid \neg\alpha \mid \alpha \vee \alpha \mid [\lambda]\alpha \tag{2}$$

where $\lambda \in \Lambda$. Conjunction and implication are defined as usual, and the *dual* $\langle\lambda\rangle$ of the operator $[\lambda]$ is defined as $\neg[\lambda]\neg$. Semantics is given as follows: for any T-coalgebra $\mathfrak{A} = (X, f)$ and formula α, the truth set $\|\alpha\|_{\mathfrak{A}} \subseteq X$ is defined by

- $\| \bot \|_{\mathfrak{A}} = \emptyset$
- $\|\neg\alpha\|_{\mathfrak{A}} = X \setminus \|\alpha\|_{\mathfrak{A}}$
- $\|\alpha \vee \beta\|_{\mathfrak{A}} = \|\alpha\|_{\mathfrak{A}} \cup \|\beta\|_{\mathfrak{A}}$
- $\|[\lambda]\alpha\|_{\mathfrak{A}} = \{v \in X \mid f(v) \in \lambda_X(\|\alpha\|_{\mathfrak{A}})\}$

To get the standard box modality for Kripke semantics, we take the predicate lifting $\kappa : \breve{\mathcal{P}} \to \breve{\mathcal{P}} \circ \mathcal{P}$ given by

$$\kappa_X(Z) = \{Z' \in \mathcal{P}X \mid Z' \subseteq Z\}$$

To get the box modality for neighborhood semantics, take the predicate lifting $\nu : \breve{\mathcal{P}} \to \breve{\mathcal{P}} \circ \mathcal{N}$ given by

$$\nu_X(Z) = \{N \in \mathcal{N}X \mid Z \in N\}$$

For $\mathcal{M}$-coalgebras, we use essentially the same predicate lifting, which we denote by $\mu : \mathcal{M} \to \breve{\mathcal{P}} \circ \mathcal{M}$.

2 Homomorphisms of Coalgebras

2.1 Λ-Homomorphisms

Any set of predicate liftings comes with a simple and natural notion of coalgebra homomorphism, defined as follows:

Definition 9. *Let Λ be a set of predicate liftings for a set functor T, and let $\mathfrak{A} = (X, f)$ and $\mathfrak{B} = (Y, g)$ be T-coalgebras. A Λ-homomorphism from $\mathfrak{A}$ to $\mathfrak{B}$ is a function $h : X \to Y$ such that, for each $u \in X$, $Z \subseteq Y$ and $\lambda \in \Lambda$:*

$$(\dagger) \quad g(h(u)) \in \lambda_Y(Z) \;\Rightarrow\; f(u) \in \lambda_X(h^{-1}[Z]) \tag{3}$$

If Λ is a singleton set $\{\lambda\}$, we speak about λ-homomorphisms rather than $\{\lambda\}$-homomorphisms. Any identity function is clearly a Λ-homomorphism from a coalgebra to itself, and it is easy to check that the composition of a pair of Λ-homomorphisms is always a Λ-homomorphism. The category of T-coalgebras and Λ-homomorphisms will be denoted by $\mathbb{C}_T^\Lambda$. For any set functor T and any set of predicate liftings Λ, we shall denote the obvious forgetful functor from the category $\mathbb{C}_T^\Lambda$ by the same letter U, so that $U : \mathbb{C}_T^\Lambda \to \mathbf{Set}$, and rely on context to disambiguate. We can immediately check that coalgebra morphisms are Λ-homomorphisms, so that $\mathbb{C}_T$ is a subcategory of $\mathbb{C}_T^\Lambda$:

Proposition 3. *Suppose T is an endofunctor on* **Set** *and Λ a set of predicate liftings for T. Then any T-coalgebra morphism is a Λ-homomorphism.*

Example 1. Suppose (X, f) and (Y, g) are $\mathcal{P}$-coalgebras and h a mapping $X \to Y$. Then h is a κ-homomorphism iff it is a relation homomorphism from (X, R_f) to (Y, R_g).

Example 2. Suppose (X, f) and (Y, g) are $\mathcal{N}$-coalgebras ($\mathcal{M}$-coalgebras) and h a mapping $X \to Y$. Then h is a ν-homomorphism (μ-homomorphism) iff it is a continuous mapping.

In the case of monotone predicate liftings it is simple to establish an elementary connection between Λ-homomorphisms and special formulas of the language $\mathcal{ML}(\Lambda)$. Here, let T denote $\neg \bot$.

Definition 10. *A formula ϕ is said to be $\forall_1$ if it is built up from $\bot$ and T by conjunctions, disjunctions and $[\lambda]$. Similarly, a formula ϕ is said to be $\exists_1$ if it is built up from $\bot$ and T by conjunctions, disjunctions and $\langle\lambda\rangle$.*

Note that the negation of an $\exists_1$ formula is $\forall_1$, and vice versa. This can be proved by a straightforward induction.

Proposition 4. *Suppose $\mathfrak{A} = (X, f)$ and $\mathfrak{B} = (Y, f)$ are two T-coalgebras and $h : X \to Y$ is a Λ-homomorphism with $h(u) = v$, where Λ is a set of monotone predicate liftings. If ϕ is an $\forall_1$-formula and $v \in \|\phi\|_{\mathfrak{B}}$, then $u \in \|\phi\|_{\mathfrak{A}}$. Dually, if ϕ is an $\exists_1$-formula then $u \in \|\phi\|_{\mathfrak{A}}$ implies $v \in \|\phi\|_{\mathfrak{B}}$.*

2.2 Products and Substructures

Given a set Λ of predicate liftings, we can give a natural definition of *products* of coalgebras:

Definition 11. *Relative to a set of predicate liftings Λ, we define a product of T-coalgebras $\mathfrak{A}$ and $\mathfrak{B}$ as a product of $\mathfrak{A}$ and $\mathfrak{B}$ in the category $\mathbb{C}_T^\Lambda$.*

Example 3. Given two Kripke models $\mathfrak{M}_1 = (X_1, R_1)$ and $\mathfrak{M}_2 = (X_2, R_2)$, the product $\mathfrak{M}_1 \times \mathfrak{M}_2 = (X_1 \times X_2, R_1 \times R_2)$ is defined by letting $X_1 \times X_2$ be the usual cartesian product of sets, and we define $R_1 \times R_2$ by setting

$$(u, v)(R_1 \times R_2)(u', v') \text{ iff } uR_1u' \text{ and } vR_2v' \tag{4}$$

The reader can check that this corresponds to a product in the category $\mathbb{C}_{\mathcal{P}}^\kappa$.

Example 4. Given two neighborhood models, i.e. two $\mathcal{N}$-coalgebras $\mathfrak{A} = (X, f)$ and $\mathfrak{B} = (Y, g)$, define the product $\mathfrak{A} \times \mathfrak{B} = (X \times Y, f \times g)$ by letting $X \times Y$ be the cartesian product and defining $f \times g$ by setting, for each $u \in X \times Y$:

$$(f \times g)(u) := \{\pi_X^{-1}[Z] \mid Z \in f(\pi_X(u))\} \cup \{\pi_Y^{-1}[Z] \mid Z \in g(\pi_Y(u))\} \tag{5}$$

where π_X and π_Y are the projections from $X \times Y$ to X and Y respectively. It is easy to check that this defines a product of $\mathfrak{A}$ and $\mathfrak{B}$ in the category $\mathbb{C}_{\mathcal{N}}^\nu$ of $\mathcal{N}$-coalgebras and ν-homomorphisms.

We can also define a natural notion of *substructure*, using the realization that the category $\mathbb{C}$ together with forgetful functor U defines a *construct*, and so we can talk about *initial morphisms* (see [1] for an introduction to these concepts):

Definition 12. *Let $(\mathbb{C}, U)$ be a construct, so that U is a forgetful functor from $\mathbb{C}$ to* **Set**. *Then a morphism $f : A \to B$ in $\mathbb{C}$ is called an initial morphism if, for any object C and any mapping $g : UC \to UA$: if there is a morphism $h : C \to B$ with $Uh = Uf \circ g$, then there is a morphism $g' : C \to A$ with $Ug' = g$.*

Given a functor T and predicate liftings Λ, we then say that a T-coalgebra (X, f) is a substructure of (Y, g) in $\mathbb{C}_T^\Lambda$ if $X \subseteq Y$ and the inclusion is an initial morphism from (X, f) to (Y, g).

Example 5. A Kripke model (X, R) is said to be a submodel of (X', R') if $X \subseteq X'$ and, for all $u, v \in X$, we have uRv iff $uR'v$. Submodels of Kripke models then correspond to substructures in $\mathbb{C}_{\mathcal{P}}^\kappa$.

Example 6. A neighborhood model (X, f) is said to be a submodel of (Y, g) if $X \subseteq Y$ and for all $u \in X$, $f(u) = \{Z \cap X \mid Z \in g(u)\}$. Submodels of neighborhood models then correspond to substructures in $\mathbb{C}_{\mathcal{N}}^\nu$.

2.3 Simulations

Simulations of coalgebras have been studied in depth by A. Thijs [10], and later from a modal logic perspective by A. Baltag [2]. These authors define simulations of coalgebras from socalled *relators*. Later, B. Jacobs and J. Hughes [4] have approached simulations using instead the notion of a *lax extension* of a functor (see [9] for a recent study of this concept).

Definition 13. *Let T be any set functor. A lax extension for T is a mapping L that assigns to each relation $R \subseteq X \times Y$ a relation $LR \subseteq TX \times TY$, and satisfies the conditions*

1. *$R \subseteq S$ implies $LR \subseteq LS$*
2. *$LR; LS \subseteq L(R; S)$*
3. *for a mapping $h : X \to Y$, $Th \subseteq Lh$*

Here $R; S$ means the relation composition of R and S.

Definition 14. *Given a lax extension L for functor T, an L-simulation from T-coalgebra (X, f) to (Y, g) is a relation $R \subseteq X \times Y$ such that uRv implies $f(u)(LR)g(v)$.*

Given a set of predicate liftings Λ for T, define a mapping L_Λ as follows: for a set X, $L_\Lambda X = TX$, and for a relation $R \subseteq X \times Y$, define the relation $L_\Lambda R$ as

$$\{(a, b) \in TX \times TY \mid \forall Z \subseteq Y \forall \lambda \in \Lambda : b \in \lambda_Y(Z) \Rightarrow a \in \lambda_X(R^{-1}[Z])\} \qquad (6)$$

where $R^{-1}[Z]$ is the set of $u \in X$ such that uRv for some $v \in Z$.

Proposition 5. *Whenever Λ is a monotone set of predicate liftings, the mapping L_Λ is a lax extension for T.*

Example 7. Let (X, f) and (Y, g) be $\mathcal{P}$-coalgebras. Then a relation S is an L_κ-simulation from (X, f) to (Y, g) iff it is a simulation from (X, R_f) to (Y, R_g) in the usual sense that, if uSv and uR_fu', then there is v' with vR_gv' and $u'Sv'$.

3 Structure of the Category $\mathbb{C}_T^\Lambda$

3.1 More on Neighborhood Models

The main purpose of this section is to establish a condition on predicate liftings Λ under which the category $\mathbb{C}_T^\Lambda$ is guaranteed to be bicomplete, i.e. have all (small) limits and colimits. It turns out that coalgebras for the neighborhood functor play an important role here. We generalize the neighborhood functor to represent multi-modal neighborhood models in the following straightforward manner (here, and from now on, we use $\triangleright$ as a special symbol for function application, so that $f \triangleright x$ is synonymous with $f(x)$):

Definition 15. *Let A be any set. Then the neighborhood functor $\mathcal{N}^A$ indexed by A is defined by setting $\mathcal{N}^A X = (\mathcal{N}X)^A$, i.e. the set of mappings from A to $\mathcal{N}X$. Given a mapping $h : X \to Y$, the mapping $\mathcal{N}^A h : (\mathcal{N}X)^A \to (\mathcal{N}Y)^A$ is defined by the following equation for $n \in (\mathcal{N}X)^A$ and $a \in A$:*

$$\mathcal{N}^A h(n) \triangleright a = \mathcal{N}h(n(a)) \tag{7}$$

Given a set X, an element of $(\mathcal{N}X)^A$ will be referred to as a *neighborhood structure* for A in X. Neighborhood structures come with a natural order of "pointwise inclusion":

Definition 16. *Let n, n' be two neighborhood structures for A in a set X. We write $n \preceq n'$ to say that, for each $a \in A$, we have $n(a) \subseteq n'(a)$.*

Proposition 6. *The collection of neighborhood structures for A in X forms a complete lattice under the order $\preceq$, in which we write meet and join as $\bigwedge$ and $\bigvee$ respectively, and for each collection C of neighborhoods and each $a \in A$ we have*

$$(\bigvee C) \triangleright a = \bigcup_{n \in C} n(a) \tag{8}$$

and

$$(\bigwedge C) \triangleright a = \bigcap_{n \in C} n(a) \tag{9}$$

Now that we have defined the indexed version $\mathcal{N}^A$ of the neighborhood functor, it is easy to modify the predicate lifting ν to give predicate liftings for $\mathcal{N}^A$:

Definition 17. *Let A be any set. Then for any $a \in A$, let ν^a be the predicate lifting for $\mathcal{N}^A$ defined by $\nu^a_X(Z) = \{n \in \mathcal{N}^A \mid Z \in n(a)\}$. Let ν^A denote the set of predicate liftings ν^a for $a \in A$.*

We get the following result, which shows that from a categorical perspective, neighborhood models are very well behaved:

Theorem 1. *For any set A, the forgetful functor $U : \mathbb{C}^{\nu^A}_{\mathcal{N}^A} \to \mathbf{Set}$ creates limits and colimits.*

Corollary 1. *The category $\mathbb{C}^{\nu^A}_{\mathcal{N}^A}$ is bicomplete.*

3.2 Mapping T-coalgebras to Neighborhood Models

The following result was observed in [6] and shows how a natural transformation from functor T to S generally allows us to define a predicate lifting for T given that a predicate lifting for S is known:

Proposition 7. *Let T, S be functors and $\alpha : T \to S$ a natural transformation. Then, for any predicate lifting λ for S, the components $\check{\mathcal{P}}\alpha_X \circ \lambda_X$ define a predicate lifting for T.*

Given a set of predicate liftings Λ for S and a natural transformation $\alpha : T \to S$, write $\lambda \star \alpha$ for the induced predicate lifting for each $\lambda \in \Lambda$ and write $\Lambda \star \alpha = \{\lambda \star \alpha \mid \lambda \in \Lambda\}$.

Given a natural transformation $\alpha : T \to S$, let the mapping F_α send a T -coalgebra $f : X \to TX$ to the S-coalgebra $\alpha_X \circ f : X \to SX$, and let F_α send a Λ-homomorphism $h : \mathfrak{A} \to \mathfrak{B}$ of T-coalgebras $\mathfrak{A}, \mathfrak{B}$ to the mapping h itself (but with domain and codomain changed to $F_\alpha\mathfrak{A}$ and $F_\alpha\mathfrak{B}$ respectively).

Proposition 8. *The mapping F_α defines a covariant functor from the category of T-coalgebras and $\Lambda \star \alpha$-homomorphisms to the category of S-coalgebras and Λ-homomorphisms.*

This proposition has an important special case.

Definition 18. *The transposite of a predicate lifting $\lambda : \breve{\mathcal{P}} \to \breve{\mathcal{P}} \circ T$ is the natural transformation $\lambda^\flat : T \to \mathcal{N}$ defined by $\lambda^\flat_X(a) = \{Z \subseteq X \mid a \in \lambda_X(Z)\}$ (see [9]). Similarly, the transposite of a set of predicate liftings Λ is the natural transformation $\Lambda^\flat : T \to \mathcal{N}^\Lambda$ defined by letting $\Lambda^\flat_X(a)$ be the neighborhood structure n for the set Λ sending each $\lambda \in \Lambda$ to $\lambda^\flat_X(a)$.*

Proposition 9. *For any set of predicate liftings Λ for T and any $\lambda \in \Lambda$, we have*

$$\nu^\lambda \star \Lambda^\flat = \lambda \tag{10}$$

Hence, given a set of predicate liftings Λ for T, the functor $F_{\Lambda^\flat}$ can be regarded as a functor from $\mathbb{C}^\Lambda_T$ to $\mathbb{C}^{\nu^\Lambda}_{\mathcal{N}^\Lambda}$. This functor sends a T-coalgebra (X, f) to the $\mathcal{N}^\Lambda$-coalgebra (X, f') defined by

$$f'(u) \triangleright \lambda = \lambda^\flat_X(f(u)) \tag{11}$$

and its action on a mapping $h : \mathfrak{A} \to \mathfrak{B}$ is defined by taking the same mapping on the underlying sets of the coalgebras and changing the domain to $F_{\Lambda^\flat}(\mathfrak{A})$ and the codomain to $F_{\Lambda^\flat}(\mathfrak{B})$. This functor will be called the *neighborhood representation functor* associated with the category $\mathbb{C}^\Lambda_T$.

3.3 Limits and Colimits in $\mathbb{C}^\Lambda_T$

We now turn to the main task of the section, to establish a condition that guarantees that the category $\mathbb{C}^\Lambda_T$ is bicomplete.

Definition 19. *Let Λ be a set of predicate liftings for functor T and let $a \in TX$ for some set X. A neighborhood structure $n : \Lambda \to \mathcal{N}X$ is said to be a basis for a if*

- $n \preceq \Lambda^\flat_X(a)$
- *for any $b \in TX$, if $n \preceq \Lambda^\flat_X(b)$ then $\Lambda^\flat_X(a) \preceq \Lambda^\flat_X(b)$*

Definition 20. *A set Λ of predicate liftings for a functor T is said to be strongly separating if, for any set X and any neighborhood structure $n : \Lambda \to \mathcal{N}X$, there is a unique $a \in TX$ such that n is a basis for a.*

We also say that a basis for *a generates a*. Note that every strongly separating set of predicate liftings Λ gives rise to a mapping $\beta_X : (\mathcal{N}X)^\Lambda \to (\mathcal{N}X)^\Lambda$ for each set X, by letting β_X send a neighborhood structure n to the neighborhood structure $\Lambda^\flat_X(a)$, where a is the member of FX generated by n. We call the assignment β of the mapping β_X to each set X the *closure operator* associated with Λ.

In the case where Λ is a singleton set $\{\lambda\}$, we can simplify matters a bit: a neighborhood structure in X is then simply taken to be a member of $\mathcal{N}X$, and the definition of strong separation is given as before but with the order $\preceq$ replaced by the subsethood relation. In the case that λ is strongly separating, we can then consider the mapping β_X for a given set X to have the type $\mathcal{N}X \to \mathcal{N}X$ instead of $(\mathcal{N}X)^{\{\lambda\}} \to (\mathcal{N}X)^{\{\lambda\}}$.

In order to establish bicompleteness of the category $\mathbb{C}^\Lambda_T$, we shall need one extra restriction on the set of predicate liftings Λ, which is essentially a sort of weak "naturality" condition on the closure operator β. For this definition we introduce the following notation:

Definition 21. *Let $h : X \to Y$ be any mapping. Then the mapping*

$$h^\dagger : (\mathcal{N}Y)^\Lambda \to (\mathcal{N}X)^\Lambda \tag{12}$$

is defined by taking a neighborhood structure $n : \Lambda \to \mathcal{N}Y$ to $n' : \Lambda \to \mathcal{N}X$ defined by

$$n'(\lambda) := \{h^{-1}[Z] \mid Z \in n(\lambda)\} \tag{13}$$

Proposition 10. *Let n, n' be any neighborhood structures in Y and let $h : X \to Y$ be any mapping. Then $h^\dagger$ is monotone in the sense that*

$$n \preceq n' \;\Rightarrow\; h^\dagger(n) \preceq h^\dagger(n') \tag{14}$$

Definition 22. *Let Λ be a strongly separating set of predicate liftings for T. We say that the closure operator associated with Λ reflects inverse images if, for every mapping $h : X \to Y$ and any neighborhood structure n in Y we have*

$$h^\dagger(\beta_Y(n)) \preceq \beta_X(h^\dagger(n)) \tag{15}$$

If Λ is a singleton set, this amounts to saying that, for every set $N \in \mathcal{N}Y$, we have

$$\breve{\mathcal{P}}h[\beta_Y(N)] \subseteq \beta_X(\breve{\mathcal{P}}h[N]) \tag{16}$$

A less compact but more transparent way of saying that the closure operator corresponding to Λ reflects inverse images in this case is that, for every neighborhood structure N in Y and all $Z \subseteq Y$,

$$Z \in \beta_Y(N) \Rightarrow h^{-1}[Z] \in \beta_X(\{h^{-1}[Z'] \mid Z' \in N\}) \tag{17}$$

This should also make the choice of terminology more apparent.

Example 8. The predicate lifting ν is strongly separating and since its closure operator just assigns the identity mapping on $\mathcal{N}X$ to any set X, it clearly reflects inverse images.

Example 9. The predicate lifting μ for $\mathcal{M}$ is strongly separating, and its closure operator β assigns to a set X the mapping β_X defined by

$$\beta_X(N) :=\uparrow N = \{Z \in \mathcal{N}X \mid Z' \subseteq Z \text{ for some } Z' \in N\} \tag{18}$$

It is easily checked that this closure operator reflects inverse images.

Example 10. The predicate lifting κ is strongly separating and its closure operator is given by

$$N \mapsto \uparrow\{\bigcap N\} \tag{19}$$

This closure operator reflects inverse images.

In the rest of this section, let Λ be a strongly separating set of predicate liftings for functor T such that its associated closure operator reflects inverse images, and denote the neighborhood representation functor simply by $F : \mathbb{C}_T^{\Lambda} \to \mathbb{C}_{\mathcal{N}^{\Lambda}}^{\nu^{\Lambda}}$. We define a functor $G : \mathbb{C}_{\mathcal{N}^{\Lambda}}^{\nu^{\Lambda}} \to \mathbb{C}_T^{\Lambda}$ as follows: given an $\mathcal{N}^{\Lambda}$-coalgebra $\mathfrak{A} = (X, f)$, define $G(\mathfrak{A}) = (X, f')$ by letting f' send each $u \in X$ to the unique $a \in TX$ generated by $f(u)$. The action by G on morphisms is defined in the obvious way, by just changing the domain and codomain of a mapping $h : \mathfrak{A} \to \mathfrak{B}$ to $G(\mathfrak{A})$ and $G(\mathfrak{B})$ respectively. That this indeed defines a covariant functor is immediate from the assumption that the closure operator reflects inverse images. Furthermore, it is easy to check the following:

Lemma 1. *The following facts hold:*

1. $U \circ F = U$ *and* $U \circ G = U$
2. G *is right adjoint to* F*, and furthermore*
3. $G \circ F$ *is the identity functor on* $\mathbb{C}_T^{\Lambda}$

We can now prove the following:

Theorem 2. *The category* $\mathbb{C}_T^{\Lambda}$ *is bicomplete.*

This result guarantees that, for any strongly separating set Λ of predicate liftings, we can find products of models in the category $\mathbb{C}_T^{\Lambda}$, and indeed any other model theoretic construction that may be represented as a limit or colimit of a suitable diagram.

Although this result subsumes some interesting special cases, the premises of the theorem are quite restrictive and an important task for future research is to look for milder conditions than the strong separation condition used here to guarantee existence of useful model theoretic constructions in the category $\mathbb{C}_T^{\Lambda}$. For example, in the case of probabilistic coalgebras (i.e. coalgebras for the functor that sends each set X to the set of probability distrubitions over X with finite support), predicate liftings are in the simplest case of the form $L(p)$ for $p \in [0,1] \cap \mathbb{Q}$ with $L(p)_X(Z)$ defined as the set of probability distributions d with $d(Z) \geq p$. Clearly this set of predicate liftings cannot be strongly separating; just take a neighborhood structure that assigns $\emptyset$ to the predicate lifting $L(1)$, and we see that this cannot be a basis for any probability distribution over X. So in this case, we cannot derive any information about the structure of the category of Λ-homomorphisms from the results in this paper.

4 A Different Approach

A different definition of coalgebra homomorphisms, which is also parametric in a set of predicate liftings, has been suggested along with a notion of simulation by Daniel Gorín and Lutz Schröder. In this section we compare these two approaches formally, and it turns out that in the case of monotone predicate liftings the two notions are in a sense each other's duals. To distinguish the approach of Gorín and Schröder from the one presented here, their homomorphisms will be called "forward-directed Λ-homomorphisms" (while they are simply called Λ-homomorphisms in [8]).

Definition 23. *A forward-directed Λ-homomorphism is a mapping $h : X \to Y$ such that, for all $u \in X$, all $\lambda \in \Lambda$ and all $Z \subseteq X$,*

$$f(u) \in \lambda_X(Z) \Rightarrow g(h(u)) \in \lambda_Y(h[Z]) \tag{20}$$

In the case of the predicate lifting κ for $\mathcal{P}$, it is clear that κ-homomorphisms are different from forward-directed κ-homomorphisms, and so they do not correspond to relational homomorphisms. However, in this case, we can remedy the situation by passing to the *dual* of the predicate lifting κ:

Definition 24. *The dual of a predicate lifting λ, denoted $\lambda^d : \breve{\mathcal{P}} \to \breve{\mathcal{P}} \circ T$, is defined by*

$$\lambda^d_X(Z) = TX \setminus (\lambda_X(X \setminus Z)) \tag{21}$$

The set $\{\lambda^d \mid \lambda \in \Lambda\}$ will be denoted by Λ^d.

Note that the dual of a monotone predicate lifting is again a monotone predicate lifting. It is simple to check that, given $\mathcal{P}$-coalgebras (X, f) and (Y, g), a mapping $h : X \to Y$ is a relational homomorphism from (X, R_f) to (Y, R_g) iff it is a forward-directed κ^d-homomorphism. More generally, whenever Λ is a monotone set of predicate liftings we can establish the following connection between the two notions of homomorphism:

Proposition 11. *Suppose Λ is a monotone set of predicate liftings, let (X, f) and (Y, g) be T-coalgebras and let $h : X \to Y$ be any mapping. Then h is a Λ-homomorphism iff it is a forward-directed Λ^d-homomorphism.*

The constraint that Λ should be a set of monotone predicate liftings is not redundant, and generally not every Λ-homomorphism is representable as a forward-directed Λ^d-homomorphism. For the purpose of demonstrating this with an example of independent interest, we consider *topological coalgebras* (see [3]):

Definition 25. *Let $\mathcal{T}$ be a topology over a set X. Then the $\mathcal{N}$-coalgebra induced by $\mathcal{T}$ is given by the transition structure $f : X \to \mathcal{N}X$ defined by*

$$u \mapsto \{Z \in \mathcal{T} \mid u \in X\} \tag{22}$$

An $\mathcal{N}$-coalgebra is said to be a topological coalgebra if it is the coalgebra induced by some topology on its domain.

It is simple to check that, given topological coalgebras (X, f) and (Y, g) induced by topologies $\mathcal{T}_X$ and $\mathcal{T}_Y$ respectively, a mapping $h : X \to Y$ is a ν-homomorphism iff it is a continuous map from $\mathcal{T}_X$ to $\mathcal{T}_Y$.

Consider the following example: let X be the set $\{u, v, w\}$ and let $Y = \{u, v\}$. Let $\mathcal{T}_X$ be the topology on X given by

$$\{\emptyset, \{w\}, \{u, v, w\}\} \tag{23}$$

Let $\mathcal{T}_Y$ be the topology on Y given by

$$\{\emptyset, \{v\}, \{u, v\}\} \tag{24}$$

Let $f : X \to \mathcal{N}X$ be the $\mathcal{N}$-coalgebra on X induced by $\mathcal{T}_X$ and $g : Y \to \mathcal{N}Y$ be the coalgebra on Y induced by $\mathcal{T}_Y$. Consider the mapping $h : X \to Y$ defined by $u \mapsto u$, $v \mapsto u$ and $w \mapsto v$. This is clearly a continuous map from $\mathcal{T}_X$ to $\mathcal{T}_Y$, but it is not a forward-directed ν^d-homomorphism. To see this, we have $f(w) \in \nu^d_X(\{u\})$, but

$$g(h(w)) = g(v) \notin \nu^d_Y(\{u\}) = \nu^d_Y(h[\{u\}]) \tag{25}$$

Indeed, there doesn't seem to be any way of representing continuous maps using forward-directed homomorphisms. On the other hand, forward-directed homomorphisms for ν (rather than its dual ν^d) have a natural interpretation in this context too: the reader can check that, given topological coalgebras (X, f) and (Y, g) induced by topologies $\mathcal{T}_X$ and $\mathcal{T}_Y$ respectively, a mapping $h : X \to Y$ is a forward-directed ν-homomorphism iff it is an open map from $\mathcal{T}_X$ to $\mathcal{T}_Y$.

References

1. Adamek, J., Herrlich, H., Strecker, G.E.: The Joy of Cats: Abstract and Concrete Categories. University of Bremen (2004)
2. Baltag, A.: A logic for coalgebraic simulation. In: Reichel, H. (ed.) Coalgebraic Methods in Computer Science, CMCS 2000. ENTCS, vol. 33, pp. 42–60. Elsevier (2000)
3. Chung, K.O.: Weak homomorphisms of coalgebras. Doctoral dissertation. Iowa State University (2007)
4. Jacobs, B., Hughes, J.: Simulations in coalgebra. Theoret. Comput. Sci. 327, 71–108 (2004)
5. Moss, L.: Coalgebraic logic. Ann. Pure Appl. Logic 96, 277–317 (1999)
6. Pattinson, D.: Translating logics for coalgebras. In: Wirsing, M., Pattinson, D., Hennicker, R. (eds.) WADT 2003. LNCS, vol. 2755, pp. 393–408. Springer, Heidelberg (2003)
7. Pattinson, D.: Coalgebraic modal logic: soundness, completeness and decidability of local consequence. Theoret. Comput. Sci. 309, 177–193 (2003)
8. Gorín, D., Schröder, L.: Simulations and bisimulations for coalgebraic modal logics. In: Heckel, R., Milius, S. (eds.) CALCO 2013. LNCS, vol. 8089, pp. 253–266. Springer, Heidelberg (2013)
9. Marti, J., Venema, Y.: Lax extensions of coalgebra functors. In: Pattinson, D., Schröder, L. (eds.) CMCS 2012. LNCS, vol. 7399, pp. 150–169. Springer, Heidelberg (2012)
10. Thijs, A.: Simulation and fixpoint semantics. Doctoral dissertation. Rijksuniversiteit Groningen (1996)

A Proofs

Proof of Theorem 1: Let $D : \mathbb{D} \to \mathbb{C}^{\nu^A}_{\mathcal{N}^A}$ be a diagram in $\mathbb{C}^{\nu^A}_{\mathcal{N}^A}$ and index the objects $D(x)$ with $x \in \mathbb{D}$ as $\{\mathfrak{A}_i\}_{i\in I}$ with each $\mathfrak{A}_i$ written as (X_i, f_i). Let $(L, \{p_i\}_{i\in I})$ be a limit of $U \circ D$ in **Set**, where U is the forgetful functor $\mathbb{C}^{\Lambda}_{F} \to$ **Set** such that for each $i \in I$ we have $p_i : L \to U(\mathfrak{A}_i)$. Now, for each $u \in L$, let n_u be the neighborhood structure

$$\bigvee_{i\in I} p_i^{\dagger}(f_i(p_i(u))) \tag{26}$$

For each $u \in L$, let $f_L(u) = n_u$. This defines a coalgebra $f_L : L \to \mathcal{N}L$, and it is straightforward to show that $((L, f_L), \{p_i\}_{i\in I})$ defines a limit of D.

We now show that U creates colimits: let $D : \mathbb{D} \to \mathbb{C}^{\nu^A}_{\mathcal{N}^A}$ be a diagram in $\mathbb{C}^{\nu^A}_{\mathcal{N}^A}$ and index the objects $D(x)$ with $x \in \mathbb{D}$ as $\{\mathfrak{A}_i\}_{i\in I}$ with each $\mathfrak{A}_i$ written as (X_i, f_i). Let $(L, \{p_i\}_{i\in I})$ be a colimit of $U \circ D$ in **Set** with $p_i : U(\mathfrak{A}_i) \to L$ for each $i \in I$. For each $u \in L$ and $i \in I$, let n_u be the neighborhood structure

$$\bigwedge_{i\in I} \bigwedge_{p_i(v)=u} \mathcal{N}^A p_i(f_i(v)) \tag{27}$$

Define $f_L : L \to \mathcal{N}L$ by setting each $f_L(u) = n_u$ for each $u \in L$. This defines an $\mathcal{N}^A$-coalgebra with domain L, and routine calculations will show that (L, f_L) together with the mappings $\{p_i\}_{i\in I}$ defines a colimit of D.

Proof of Proposition 8: Let $f : X \to TX$ and $g : Y \to TY$ be T-coalgebras and $h : X \to Y$ a $\Lambda \star \alpha$-homomorphism. Let $u \in X$, $Z \subseteq Y$ and $\lambda \in \Lambda$. Then we have

$$\begin{aligned}
\alpha_Y \circ g(h(u)) \in \lambda_Y(Z) &\Leftrightarrow g(h(u)) \in \breve{\mathcal{P}}\alpha_Y \circ \lambda_Y(Z) \\
&\Leftrightarrow g(h(u)) \in (\lambda \star \alpha)_Y(Z) \\
&\Rightarrow f(u) \in (\lambda \star \alpha)_X(h^{-1}[Z]) \\
&\Leftrightarrow f(u) \in \breve{\mathcal{P}}\alpha_X \circ \lambda_X(h^{-1}[Z]) \\
&\Leftrightarrow \alpha_X \circ f(u) \in \lambda_X(h^{-1}[Z])
\end{aligned}$$

and this shows that h is a Λ-homomorphism from $\alpha_X \circ f : X \to SX$ to $\alpha_Y \circ g : Y \to SY$ as well.

Proof of Proposition 9: We need to show that, for any set X, we have

$$\breve{\mathcal{P}}\Lambda^{\flat}_X \circ \nu^{\lambda}_X = \lambda_X \tag{28}$$

Given $Z \subseteq X$, we have

$$\begin{aligned}
\breve{\mathcal{P}}\Lambda^{\flat}_X \circ \nu^{\lambda}_X(Z) &= \{a \in TX \mid \Lambda^{\flat}_X(a) \in \nu^{\lambda}_X(Z)\} \\
&= \{a \in TX \mid Z \in \lambda^{\flat}_X(a)\} \\
&= \{a \in TX \mid a \in \lambda_X(Z)\} \\
&= \lambda_X(Z)
\end{aligned}$$

Proof of Theorem 2: Completeness is a simple consequence of preservation of limits by right adjoints, together with the fact that $G \circ F$ is the identity functor on $\mathbb{C}_T^{\Lambda}$.

We now show that $\mathbb{C}_T^{\Lambda}$ has all small colimits. Let $D : \mathbb{D} \to \mathbb{C}_T^{\Lambda}$ be a small diagram in $\mathbb{C}_T^{\Lambda}$, and index the objects Dx whith $x \in obj(\mathbb{D})$ as $\{\mathfrak{A}_i\}_{i \in I}$. Then we get a diagram $F \circ D : \mathbb{D} \to \mathbb{C}_{\mathcal{N}^{\Lambda}}^{\nu^{\Lambda}}$, and since the forgetful functor $U : \mathbb{C}_{\mathcal{N}^{\Lambda}}^{\nu^{\Lambda}} \to \mathbf{Set}$ creates colimits we find a colimit $(\mathfrak{L}, \{h_i\}_{i \in I})$ of $F \circ D$, $h_i : \mathfrak{A}_i \to \mathfrak{L}$, such that $(U\mathfrak{L}, \{Uh_i\}_{i \in I})$ is a colimit of $U \circ F \circ D = U \circ D$ in $\mathbf{Set}$. Then $(G\mathfrak{L}, \{Gh_i\}_{i \in I})$ is a cocone for the diagram $G \circ F \circ D = D$ in $\mathbb{C}_T^{\Lambda}$, and we are going to show that it is in fact a colimit.

Let $(\mathfrak{B}, \{p_i\}_{i \in I})$ be a cocone for D in $\mathbb{C}_T^{\Lambda}$. We need to find a unique connecting map $c_{\mathfrak{B}} : G\mathfrak{L} \to \mathfrak{B}$ such that $p_i = c_{\mathfrak{B}} \circ Gh_i$ for each $i \in I$. Now, first note that $(F\mathfrak{B}, \{F(p_i)\}_{i \in I})$ is a cocone for $F \circ D$, and so there is a unique connecting map $c_{F\mathfrak{B}} : \mathfrak{L} \to F(\mathfrak{B})$ with $Fp_i = c_{F(\mathfrak{B})} \circ h_i$ for each $i \in I$. Then we get a morphism

$$G(c_{F(\mathfrak{B})}) : G\mathfrak{L} \to GF\mathfrak{B} = \mathfrak{B} \tag{29}$$

and it is a connecting map since for each $i \in I$ we have

$$Fp_i = c_{F(\mathfrak{B})} \circ h_i \tag{30}$$

and so

$$p_i = GFp_i) = G(c_{F(\mathfrak{B})}) \circ Gh_i \tag{31}$$

It now suffices to show uniqueness of this connecting map. Suppose c and c' are both connecting maps. First, note that since

$$(UG\mathfrak{L}, \{UGh_i\}_{i \in I}) = (U\mathfrak{L}, \{Uh_i\}_{i \in I}) \tag{32}$$

we get that $(UG\mathfrak{L}, \{UGh_i\}_{i \in I})$ is a colimit of $U \circ D = U \circ F \circ D$ in $\mathbf{Set}$. Furthermore, Uc and Uc' are both connecting maps from the colimit $(UG\mathfrak{L}, \{UGh_i\}_{i \in I})$ to the cocone $(U\mathfrak{B}, \{Up_i\}_{i \in I})$ for $U \circ D$. So $Uc = Uc'$, and since the forgetful functor $U : \mathbb{C}_T^{\Lambda} \to \mathbf{Set}$ is clearly faithful we get $c = c'$ as required.

Proof of Proposition 11: Suppose that h is a Λ-homomorphism. Let $Z \subseteq X$, $u \in X$ and $\lambda^d \in \Lambda^d$ be such that $f(u) \in \lambda_X^d(Z)$. We have

$$\begin{aligned} g(h(u)) \notin \lambda_Y^d(h[Z]) &\Leftrightarrow g(h(u)) \in \lambda_Y(Y \setminus h[Z]) \\ &\Rightarrow f(u) \in \lambda_X(h^{-1}[Y \setminus h[Z]]) \\ &\Rightarrow f(u) \in \lambda_X(X \setminus Z) \\ &\Leftrightarrow f(u) \notin \lambda_X^d(Z) \end{aligned}$$

so we must have $g(h(u)) \in \lambda_Y^d(h[Z])$, and we see that h is a forward-directed Λ^d-homomorphism. We have tacitly used the fact that $h^{-1}[Y \setminus h[Z]] \subseteq X \setminus Z$.

Conversely, suppose h is a forward-directed Λ^d-homomorphism. Let $Z \subseteq Y$, $u \in X$ and $\lambda \in \Lambda$ be such that $g(h(u)) \in \lambda_Y(Z)$. We have

$$\begin{aligned} f(u) \notin \lambda_X(h^{-1}[Z]) &\Leftrightarrow f(u) \in \lambda^d_X(X \setminus h^{-1}[Z]) \\ &\Leftrightarrow f(u) \in \lambda^d_X(h^{-1}[Y \setminus Z]) \\ &\Rightarrow g(h(u)) \in \lambda^d_Y(h[h^{-1}[Y \setminus Z]]) \\ &\Rightarrow g(h(u)) \in \lambda^d_Y(Y \setminus Z) \\ &\Leftrightarrow g(h(u)) \notin \lambda_Y(Z) \end{aligned}$$

and so we must have $f(u) \in \lambda_X(h^{-1}[Z])$, so h is a Λ-homomorphism. We have used here the fact that $h[h^{-1}[Y \setminus Z]] \subseteq Y \setminus Z$ and that each λ^d is monotone.

From Kleisli Categories to Commutative C^*-Algebras: Probabilistic Gelfand Duality

Robert Furber and Bart Jacobs

Institute for Computing and Information Sciences (iCIS),
Radboud University Nijmegen, The Netherlands
www.cs.ru.nl/B.Jacobs

Abstract. C^*-algebras form rather general and rich mathematical structures that can be studied with different morphisms (preserving multiplication, or not), and with different properties (commutative, or not). These various options can be used to incorporate various styles of computation (set-theoretic, probabilistic, quantum) inside categories of C^*-algebras. This paper concentrates on the commutative case and shows that there are functors from several Kleisli categories, of monads that are relevant to model probabilistic computations, to categories of C^*-algebras. This yields a new probabilistic version of Gelfand duality, involving the "Radon" monad on the category of compact Hausdorff spaces. We also show that a commutative C^*-algebra is isomorphic to the space of convex continuous functionals from its state space to the complex numbers. This allows us to obtain an appropriately commuting state-and-effect triangle for commutative C^*-algebras.

1 Introduction

There are several notions of computation. We have the classical notion of computation, probabilistic computation, where a computer may make random choices, and quantum computation, which uses quantum mechanical interference and measurement. Normally we would consider classical computation to be done on sets, probabilistic computation on spaces with a measure, and quantum computation on Hilbert spaces. We can instead use categories with C^*-algebras as objects and a choice of either *-homomorphisms (called MIU-map below) or positive unital maps as the morphisms. The general outline is represented in this table.

	set-theoretic	**probabilistic**	**quantum**
C^*-algebras	commutative	commutative	non-commutative
maps preserve	multiplication involution unit	positivity unit	positivity unit
maps abbreviation	MIU	PU	PU

R. Heckel and S. Milius (Eds.): CALCO 2013, LNCS 8089, pp. 141–157, 2013.

While the quantum case is an important source of motivation, we will be concerned with the classical and probabilistic cases in this article. In particular, we will relate the alternative method of representing probabilistic computation, using monads, to the C^*-algebraic approach.

In recent years the methods and tools of category theory have been applied to Hilbert spaces — see *e.g.* [1] and the references there — and also to C^*-algebras, see for instance [23,21]. In this paper we show that clearly distinguishing different types of homomorphisms of C^*-algebras already brings quite some clarity. Moreover, we demonstrate the relevance of monads (and their Kleisli and Eilenberg-Moore categories) in this field. The aforementioned paper [23] concerns itself with only the *-homomorphisms (*i.e.* with the MIU-maps in our terminology).

Giry [10, I.4] described how we can consider a stochastic process as being a diagram in the Kleisli category of the Giry monad on measure spaces. By using the Radon monad on compact spaces instead, we can get a different category of stochastic processes on compact spaces as diagrams in the (opposite of the) category of *commutative* C^*-algebras with PU-maps. This allows the quantum generalization to taking diagrams in the category of *non-commutative* C^*-algebras. The relationship to quantum computation is that $B(\mathcal{H})$, the algebra of all bounded operators on a Hilbert space is a C^*-algebra, and for every C^*-algebra A, there is a Hilbert space $\mathcal{H}$ such that A is isomorphic to a norm-closed *-subalgebra of $B(\mathcal{H})$. The category of C^*-algebras allows us to represent measurement with maps from a commutative C^*-algebra to $B(\mathcal{H})$. We can also represent composite systems that are partly quantum and partly classical. Girard also used certain special C^*-algebras, von Neumann algebras, for his Geometry of Interaction [9].

2 Preliminaries on C^*-Algebras

We write $\mathbf{Vect} = \mathbf{Vect}_{\mathbb{C}}$ for the category of vector spaces over the complex numbers $\mathbb{C}$. This category has direct product $V \oplus W$, forming a biproduct (both a product and a coproduct) and tensors $V \otimes W$, which distribute over $\oplus$. The tensor unit is the space $\mathbb{C}$ of complex numbers. The unit for $\oplus$ is the singleton (null) space 0. We write $\overline{V}$ for the vector space with the same vectors/elements as V, but with conjugate scalar product: $z \bullet_{\overline{V}} v = \overline{z} \bullet_V v$. This makes **Vect** an involutive category, see [14].

A **-algebra* is an involutive monoid A in the category **Vect**. Thus, A is itself a vector space, carries a multiplication $\cdot\colon A \otimes A \to A$, linear in each argument, and has a unit $1 \in A$. Moreover, there is an involution map $(-)^*\colon \overline{A} \to A$, preserving 0 and $+$ and satisfying:

$$1^* = 1 \qquad (x \cdot y)^* = y^* \cdot x^* \qquad x^{**} = x \qquad (z \bullet x)^* = \overline{z} \bullet x^*.$$

Here we have written a fat dot $\bullet$ for scalar multiplication, to distinguish it from the algebra's multiplication $\cdot$. For $z = a + bi \in \mathbb{C}$ we have the conjugate $\overline{z} = a - bi$. Often we omit the multiplication dot $\cdot$ and simply write xy for $x \cdot y$. Similarly, the scalar multiplication $\bullet$ is often omitted. We then rely on the context to distinguish the two multiplications.

A *C^*-algebra* is a *-algebra A with a norm $\|-\| : A \to \mathbb{R}_{\geq 0}$ in which it is complete. This norm satisfies $\|x\| = 0$ iff $x = 0$ and:

$$\|x + y\| \leq \|x\| + \|y\| \qquad \|z \bullet x\| = |z| \cdot \|x\|$$
$$\|x \cdot y\| \leq \|x\| \cdot \|y\| \qquad \|x^* \cdot x\| = \|x\|^2.$$

Especially this last equation $\|x^* \cdot x\| = \|x\|^2$ is vital for C^*-algebras, as it distinguishes them from Banach *-algebras. In the current setting, each C^*-algebra is unital, *i.e.* has a (multiplicative) unit 1. A C^*-algebra is called *commutative* if its multiplication is commutative, and *finite-dimensional* is it has finite dimension when considered as a vector space.

An element x in a C^*-algebra A is called *positive* if it can be written in the form $x = y^* \cdot y$. We write $A^+ \subseteq A$ for the subset of positive elements in A. This subset is a cone, which is to say it is closed under addition and scalar multiplication with positive real numbers. The multiplication $x \cdot y$ of two positive elements need not be positive in general (think of matrices). The square $x^2 = x \cdot x$ of a self-adjoint element $x = x^*$, however, is obviously positive. In a *commutative* C^*-algebra the positive elements are closed under multiplication. A cone A^+ in a vector space defines a partial order as follows.

$$x \leq y \Leftrightarrow y - x \in P \tag{1}$$

This is defines an order on every C^*-algebra.

There are mainly two options when it comes to maps between C^*-algebras. The difference between them plays an important role in this paper.

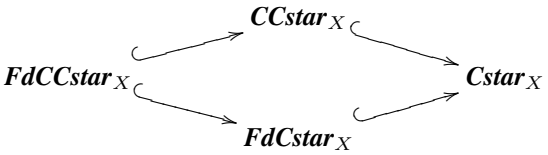
Definition 1. *We define two categories* ***Cstar***$_{MIU}$ *and* ***Cstar***$_{PU}$ *with C^*-algebras as objects, but with different morphisms.*

1. *A morphism $f : A \to B$ in* ***Cstar***$_{MIU}$ *is a linear map preserving multiplication (M), involution (I), and unit (U). Explicitly, this means for all $x, y \in A$,*

$$f(x \cdot y) = f(x) \cdot f(y) \qquad f(x^*) = f(x)^* \qquad f(1) = 1.$$

   *Often such "MIU" maps are called *-homomorphisms.*
2. *A morphism $f : A \to B$ in* ***Cstar***$_{PU}$ *is a linear map that preserves positive elements and the unit. This means that f restricts to a function $A^+ \to B^+$. Alternatively, for each $x \in A$ there is an $y \in B$ with $f(x^*x) = y^*y$.*

For both $X =$ MIU and $X =$ PU there are obvious full subcategories of commutative and/or finite-dimensional C^*-algebras, as described in:



Clearly, each "MIU" map is also a "PU" map, so that we have inclusions $\mathbf{Cstar}_{\mathrm{MIU}} \hookrightarrow \mathbf{Cstar}_{\mathrm{PU}}$, also for the various subcategories. A map that preserves positive elements is called positive itself; and a unit preserving map is called unital.

For a category $\mathbf{B}$ one often writes $\mathbf{B}(X, Y)$ or $\mathit{Hom}(X, Y)$ for the "homset" of morphisms $X \to Y$ in $\mathbf{B}$. For C^*-algebras A, B we write $\mathit{Hom}_{\mathrm{MIU}}(A, B) = \mathbf{Cstar}_{\mathrm{MIU}}(A, B)$ and $\mathit{Hom}_{\mathrm{PU}}(A, B) = \mathbf{Cstar}_{\mathrm{PU}}(A, B)$ for the homsets of MIU- and PU-maps. There is also the commonly used notion of completely positive maps, which is a stronger condition than positivity but weaker than being MIU. These maps are important when defining the tensor of C^*-algebras as a functor, as the tensor of positive maps need not be positive. They are also widely considered to represent the physically realizable transformations. Positive, but non-completely positive maps of C^*-algebras also have their uses, as entanglement witnesses for example[12, theorem2]. Since we mainly consider the commutative case, where positive and completely positive coincide, we do not consider the category of C^*-algebras with completely positive maps any further in this paper.

We collect some basic (standard) properties of PU-morphisms between C^*-algebras (see *e.g.* [25,3]).

Lemma 1. *A PU-map, i.e. a morphism in the category $\mathbf{Cstar}_{PU}$, commutes with involution $(-)^*$, and preserves the partial order $\leq$ on self-adjoint elements given by* (1).

Moreover, a PU-map f satisfies $\|f(x)\| \leq 4\|x\|$, so that $\|f(x) - f(y)\| \leq 4\|x - y\|$, making f continuous.

Proof. An element x is called self-adjoint if $x^* = x$. Each self-adjoint x can be written as difference $x = x_p - x_n$ of positive elements x_p, x_n, with $\|x_p\|, \|x_n\| \leq \|x\|$, see [5, 1.5.7]; as a result $f(x^*) = f(x) = f(x)^*$, for a PU-map f. Next, an arbitrary element y can be written as $y = y_r + iy_i$ for self-adjoint elements $y_r = \frac{1}{2}(y + y^*), y_i = \frac{1}{2i}(y - y^*)$, so that $\|y_r\|, \|y_i\| \leq \|y\|$. Then $f(y^*) = f(y)^*$. Preservation of the order is trivial.

For positive x we have $x \leq \|x\| \bullet 1$, and thus $f(x) \leq \|x\| \bullet 1$, which gives $\|f(x)\| \leq \|x\|$. An arbitrary element x can be written as linear combination of four positive elements x_i, as in $x = x_1 - x_2 + ix_3 - ix_4$, with $\|x_i\| \leq \|x\|$. Finally, $\|f(x)\| = \|f(x_1) - f(x_2) + if(x_3) - if(x_4)\| \leq \sum_i \|f(x_i)\| \leq \sum_i \|x_i\| \leq 4\|x\|$. □

The following famous result is known as Gelfand duality, relating compact Hausdorff spaces and *commutative* C^*-algebras. Notice that this result involves the "MIU" maps.

Theorem 1. *Let $\mathbf{CH}$ be the category of compact Hausdorff spaces, with continuous maps between them. Sending $X \in \mathbf{CH}$ to the algebra of continuous functions $X \to \mathbb{C}$ yields an equivalence of categories $C(-) = \mathit{Cont}(-, \mathbb{C}) \colon \mathbf{CH} \xrightarrow{\simeq} (\mathbf{CCstar}_{MIU})^{op}$.* □

The inverse to the functor $C(-)$ sends a commutative C^*-algebra A to its spectrum $\mathrm{Spec}(A)$, given by the MIU-maps $A \to \mathbb{C}$, or equivalently, by the so-called pure states (see below).

Corollary 1. *For each finite-dimensional commutative C^*-algebra A there is an $n \in \mathbb{N}$ with $A \cong \mathbb{C}^n$ in $\mathbf{FdCCstar}_{MIU}$.*

Proof. By the previous theorem there is a compact Hausdorff space X such that A is MIU-isomorphic to the algebra of continuous maps $X \to \mathbb{C}$. This X must be finite, and

since a finite Hausdorff space is discrete, all maps $X \to \mathbb{C}$ are continuous. Let $n \in \mathbb{N}$ be the number of elements in X; then we have an isomorphism $A \cong \mathbb{C}^n$. □

As we can already see in the above theorem, it is the *opposite* of a category of C^*-algebras that provides the most natural setting for computations. This is in line with what is often called Heisenberg's picture. In a logical setting it corresponds to computation of weakest preconditions, going backwards. The situation may be compared to the category of complete Heyting algebras, which is most usefully known in opposite form, as the category of locales, see [18].

For a C^*-algebra A a *state* is positive unital map $A \to \mathbb{C}$. The set $\mathit{Hom}_{\mathrm{PU}}(A, \mathbb{C})$ of such states can be equipped with the weak *-topology, which is the coarsest (smallest) topology in which all maps $\mathit{ev}_x = \lambda s.\, s(x) \colon \mathit{Hom}_{\mathrm{PU}}(A, \mathbb{C}) \to \mathbb{C}$, for $x \in A$, are continuous.

Proposition 1. *For each C^*-algebra A, the set of states $\mathit{Hom}_{PU}(A, \mathbb{C})$ is convex, and compact Hausdorff in the weak-* topology. Each PU-map $f \colon A \to B$ yields an affine continuous function $(-) \circ f \colon \mathit{Hom}_{PU}(B, \mathbb{C}) \to \mathit{Hom}_{PU}(A, \mathbb{C})$.*

We recall that a function (between convex sets) is called *affine* if it preserves convex sums. We will see shortly that such affine maps are homomorphisms of Eilenberg-Moore algebras for the distribution monad $\mathcal{D}$.

Proof. For each finite set $h_i \in \mathit{Hom}_{\mathrm{PU}}(A, \mathbb{C})$ with $r_i \in [0, 1]$ satisfying $\sum_i r_i = 1$, the function $h = \sum_i r_i h_i$ is again a state. Moreover, such convex sums are preserved by precomposition, making the maps $(-) \circ f$ affine.

If states $h, k \in \mathit{Hom}_{\mathrm{PU}}(A, \mathbb{C})$ are not equal, say $h(x) \neq k(x)$, then either the real or imaginary parts of $h(x)$ and $k(x)$ differ. Let's consider the former. Then there is a real number r with $\mathrm{re}(h(x)) < r < \mathrm{re}(k(x))$. The two open subsets $\mathit{ev}_x^{-1}(\{z \in \mathbb{C} \mid \mathrm{re}(z) < r\})$ and $\mathit{ev}_x^{-1}(\{z \in \mathbb{C} \mid \mathrm{re}(z) > r\})$ then separate h, k. Hence $\mathit{Hom}_{\mathrm{PU}}(A, \mathbb{C})$ is Hausdorff. It is compact by Alaoglu's Theorem.

Precomposition $(-) \circ f$ is continuous, since for $x \in A$ and $U \subseteq \mathbb{C}$ open we get an open subset $\big((-) \circ f\big)^{-1}(\mathit{ev}_x^{-1}(U)) = \{h \mid \mathit{ev}_x(h \circ f) \in U\} = \mathit{ev}_{f(x)}^{-1}(U)$. □

2.1 Effect Modules

Effect algebras have been introduced in mathematical physics [7], in the investigation of quantum probability, see [6] for an overview. An *effect algebra* is a partial commutative monoid $(M, 0, \mathbin{\circledcirc})$ with an orthocomplement $(-)^\perp$. One writes $x \perp y$ if $x \mathbin{\circledcirc} y$ is defined. The formulation of the commutativity and associativity requirements is a bit involved, but essentially straightforward. The orthocomplement satisfies $x^{\perp\perp} = x$ and $x \mathbin{\circledcirc} x^\perp = 1$, where $1 = 0^\perp$. There is always a partial order, given by $x \leq y$ iff $x \mathbin{\circledcirc} z = y$, for some z. The main example is the unit interval $[0, 1] \subseteq \mathbb{R}$, where addition $+$ is obviously partial, commutative, associative, and has 0 as unit; moreover, the orthocomplement is $r^\perp = 1 - r$. We write **EA** for the category of effect algebras, with morphism preserving $\mathbin{\circledcirc}$ and 1 — and thus all other structure.

For each set X, the set $[0, 1]^X$ of fuzzy predicates on X is an effect algebra, via pointwise operations. Each Boolean algebra B is an effect algebra with $x \perp y$ iff $x \wedge y = \bot$;

then $x \mathbin{⦶} y = x \vee y$. In a quantum setting, the main example is the set of effects $\mathcal{E}f(H) = \{E \colon H \to H \mid 0 \leq E \leq I\}$ on a Hilbert space H, see *e.g.* [6,11].

An *effect module* is an "effect" version of a vector space. It involves an effect algebra M with a scalar multiplication $s \bullet x \in M$, where $s \in [0,1]$ and $x \in M$. This scalar multiplication is required to be a suitable homomorphism in each variable separately. The algebras $[0,1]^X$ and $\mathcal{E}f(H)$ are clearly such effect modules. In the subcategory **EMod** $\hookrightarrow$ **EA** maps additionally commute with scalar multiplication.

For a C^*-algebra A the subset $A^+ \hookrightarrow A$ of positive elements carries a partial order $\leq$ defined on self-adjoint elements in (1). We write $[0,1]_A \subseteq A^+ \subseteq A$ for the subset of positive elements below the unit. The elements in $[0,1]_A$ will be called effects (or sometimes also: predicates). For instance, for the C^*-algebra $B(\mathcal{H})$ of bounded operators on a Hilbert space $\mathcal{H}$ the unit interval $[0,1]_{B(\mathcal{H})} \subseteq B(H)$ contains the effects $\mathcal{E}f(\mathcal{H}) = \{A \in B(\mathcal{H}) \mid 0 \leq A \leq \mathrm{id}\}$ on $\mathcal{H}$.

We claim that $[0,1]_A$ is an *effect algebra* and carries a $[0,1] \subseteq \mathbb{R}$ scalar multiplication, thus making it an *effect module*.

- Since A with $0, +$ is a partially ordered Abelian group, $[0,1]_A$ is a so-called interval effect algebra, with $x \perp y$ iff $x + y \leq 1$, and in that case $x \mathbin{⦶} y = x + y$. The ortocomplement $x^\perp$ is given by $1 - x$.
- For $r \in [0,1]$ and $x \in [0,1]_A$ the scalar multiplications rx and $(1-r)x$ are positive, and their sum is $x \leq 1$. Hence $rx \leq 1$ and thus $rx \in [0,1]_A$.

Each map of C^*-algebras $f \colon A \to B$ preserves $\leq$ and thus restricts to $[0,1]_A \to [0,1]_B$. This restriction is a map of effect modules. Hence we get a "predicate" functor $\mathbf{Cstar}_{\mathrm{PU}} \to \mathbf{EMod}$.

Lemma 2. *The functor* $[0,1]_{(-)} \colon \boldsymbol{Cstar}_{PU} \to \boldsymbol{EMod}$ *is full and faithful.*

Proof. Any PU-map $f \colon A \to B$ is completely determined (and defined by) its action on $[0,1]_A$: for a non-zero positive element $x \in A$ we use $x \leq \|x\| \, 1$ and thus $\frac{1}{\|x\|} \, x \in [0,1]_A$ to see that $f(x) = \|x\| \, f(\frac{1}{\|x\|} \, x)$. An arbitrary element $y \in A$ can be written as linear sum of four positive elements (see Lemma 1), determining $f(y)$. □

The (finite, discrete probability) distribution monad $\mathcal{D} \colon \mathbf{Sets} \to \mathbf{Sets}$ sends a set X to the set $\mathcal{D}(X) = \{\varphi \colon X \to [0,1] \mid \mathit{supp}(\varphi)$ is finite, and $\sum_x \varphi(x) = 1\}$, where $\mathit{supp}(\varphi) = \{x \mid \varphi(x) \neq 0\}$. Such an element $\varphi \in \mathcal{D}(X)$ may be identified with a finite, formal convex sum $\sum_i r_i x_i$ with $x_i \in X$ and $r_i \in [0,1]$ satisfying $\sum_i r_i = 1$. The unit $\eta \colon X \to \mathcal{D}(X)$ and multiplication $\mu \colon \mathcal{D}^2(X) \to \mathcal{D}(X)$ of this monad are given by singleton/Dirac convex sum and by matrix multiplication:

$$\eta(x) = 1x \qquad \mu(\Phi)(x) = \textstyle\sum_\varphi \Phi(\varphi) \cdot \varphi(x).$$

A *convex* set is an Eilenberg-Moore algebra of this monad: it consists of a carrier set X in which actual sums $\sum_i r_i x_i \in X$ exist for all convex combinations. We write $\mathbf{Conv} = \mathcal{EM}(\mathcal{D})$ for the category of convex sets, with "affine" functions preserving convex sums.

Effect modules and convex sets are related via a basic adjunction [17], obtained by "homming into $[0,1]$", as in:

$$\mathbf{EMod}^{\mathrm{op}} \underset{\mathbf{Conv}(-,[0,1])}{\overset{\mathbf{EMod}(-,[0,1])}{\rightleftarrows}} \mathbf{Conv} \qquad (2)$$

3 Set-Theoretic Computations in C^*-Algebras

For a set X, a function $f\colon X \to \mathbb{C}$ is called *bounded* if $|f(x)| \leq s$, for some $s \in \mathbb{R}_{\geq 0}$. We write $\ell^\infty(X)$ for the set of such bounded functions. Notice that if X is finite, any function $X \to \mathbb{C}$ is bounded, so that $\ell^\infty(X) = \mathbb{C}^X$.

Each $\ell^\infty(X)$ is a commutative C^*-algebra, with pointwise addition, multiplication and involution, and with the supremum norm $\|f\| = \inf\{s \in \mathbb{R}_{\geq 0} \mid \forall x.\, |f(x)| \leq s\}$. In fact it is a typical example of a commutative W^*-algebra, but we do not require this fact. This yields a functor $\ell^\infty\colon \mathbf{Sets} \to (\mathbf{CCstar}_{\mathrm{MIU}})^{\mathrm{op}}$, where for $h\colon X \to Y$ we have $\ell^\infty(h) = (-) \circ h\colon \ell^\infty(Y) \to \ell^\infty(X)$; it preserves the (pointwise) operations. When we restrict to the full subcategory $\mathbf{FinSets} \hookrightarrow \mathbf{Sets}$ we obtain a functor $\ell^\infty = \mathbb{C}^{(-)}\colon \mathbf{FinSets} \to (\mathbf{FdCCstar}_{\mathrm{MIU}})^{\mathrm{op}}$. The next result is then a well-known special case of Gelfand duality (Theorem 1). We elaborate the proof in some detail because it is important to see where the preservation of multiplication plays a role.

Proposition 2. *The functor $\mathbb{C}^{(-)}\colon \boldsymbol{FinSets} \to (\boldsymbol{FdCCstar}_{MIU})^{op}$ is an equivalence of categories.*

Proof. It is easy to see that the functor $\mathbb{C}^{(-)}$ is faithful. The crucial part is to see that it is full. So assume we have two finite sets, seen as natural numbers n, m, and a MIU-homomorphism $h\colon \mathbb{C}^m \to \mathbb{C}^n$. For $j \in m$, let $|\,j\rangle \in \mathbb{C}^m$ be the standard base vector with 1 at the j-th position and 0 elsewhere. Since this $|\,j\rangle$ is positive, so is $h(|\,j\rangle)$, and thus we may write it as $h(|\,j\rangle) = (r_{1j}, \ldots, r_{nj})$, with $r_{ij} \in \mathbb{R}_{\geq 0}$. Because $|\,j\rangle \cdot |\,j\rangle = |\,j\rangle$, and h preserves multiplication, we get $h(|\,j\rangle)\cdot h(|\,j\rangle) = h(|\,j\rangle)$, and thus $r_{ij}^2 = r_{ij}$. This means $r_{ij} \in \{0,1\}$, so that h is a (binary) Boolean matrix. But h is also unital, and so:

$$1 = h(1) = h(|\,1\rangle + \cdots + |\,m\rangle) = h(|\,1\rangle) + \cdots + h(|\,m\rangle). \qquad (3)$$

For each $i \in n$ there is thus precisely one $j \in m$ with $r_{ij} = 1$ — so that h is a "functional" Boolean matrix. This yields the required function $f\colon n \to m$ with $\mathbb{C}^f = h$.

Corollary 1 says that the functor $\mathbb{C}^{(-)}\colon \mathbf{FinSets} \to (\mathbf{FdCCstar}_{\mathrm{MIU}})^{\mathrm{op}}$ is essentially surjective on objects, and thus an equivalence. □

This proof demonstrates that preservation of multiplication, as required for "MIU" maps, is a rather strong condition. We make this more explicit.

Corollary 2. *For $n \in \mathbb{N}$ we have $Hom_{MIU}(\mathbb{C}^n, \mathbb{C}) \cong n$.*

Proof. When we identify $n \in \mathbb{N}$ with the n-element set $n = \{0, 1, \ldots, n-1\} \in \mathbf{FinSets}$, we get by Proposition 2, $Hom_{\mathrm{MIU}}(\mathbb{C}^n, \mathbb{C}) \cong \mathbf{FinSets}(1, n) \cong n$. □

4 Discrete Probabilistic Computations in C^*-Algebras

We turn to probabilistic computations and will see that we remain in the world of commutative C^*-algebras, but with PU-maps (positive unital) instead of MIU-maps. Recall that states of a C^*-algebra A are the PU-maps $A \to \mathbb{C}$.

Lemma 3. *Sending a set X to the set of states of the C^*-algebra $\ell^\infty(X)$ yields the expectation monad $\mathcal{E}$ from [16]: the mapping $X \mapsto \mathit{Hom}_{PU}(\ell^\infty(X), \mathbb{C})$ is isomorphic to the expectation monad $\mathcal{E}\colon \mathbf{Sets} \to \mathbf{Sets}$, defined in [16] via effect module homomorphisms: $\mathcal{E}(X) = \mathbf{EMod}\big([0,1]^X, [0,1]\big)$.*

As a result, $\mathit{Hom}_{PU}(\mathbb{C}^n, \mathbb{C}) \cong \mathcal{D}(n)$, for $n \in \mathbb{N}$, where $\mathcal{D}(n)$ is the standard n-simplex.

Proof. The predicate/effect functor $[0,1]_{(-)}\colon \mathbf{Cstar}_{\mathrm{PU}} \to \mathbf{EMod}$ is full and faithful by Lemma 2, and so:

$$\mathit{Hom}_{\mathrm{PU}}\big(\ell^\infty(X), \mathbb{C}\big) \cong \mathbf{EMod}\big([0,1]_{\ell^\infty(X)}, [0,1]_{\mathbb{C}}\big) = \mathbf{EMod}\big([0,1]^X, [0,1]\big).$$

The isomorphism $\alpha\colon \mathit{Hom}_{\mathrm{PU}}(\mathbb{C}^n, \mathbb{C}) \xrightarrow{\cong} \mathcal{D}(n)$ follows because the expectation and distribution monad coincide on finite sets, see [16]. Explicitly, it is given by $\alpha(h) = \lambda i \in n.\, h(|i\rangle)$ and $\alpha^{-1}(\varphi)(v) = \sum_i \varphi(i) \cdot v(i)$. □

The unit η and multiplication μ structure on $\mathcal{E}(X) \cong \mathit{Hom}_{\mathrm{PU}}(\ell^\infty(X), \mathbb{C})$ is very much like for "continuation" or "double dual" monads, see [19,22,13], with:

$$\begin{array}{ccc} X \xrightarrow{\eta} \mathit{Hom}_{\mathrm{PU}}(\ell^\infty(X), \mathbb{C}) & \quad & \mathit{Hom}_{\mathrm{PU}}\Big(\ell^\infty\big(\mathit{Hom}_{\mathrm{PU}}(\mathbb{C}^X, \mathbb{C})\big), \mathbb{C}\Big) \xrightarrow{\mu} \mathit{Hom}_{\mathrm{PU}}(\ell^\infty(X), \mathbb{C}) \\ x \longmapsto \lambda v.\, v(x) & & g \longmapsto \lambda v.\, g\big(\lambda h.\, h(v)\big). \end{array}$$

For an arbitrary monad $T = (T, \eta, \mu)$ on a category $\mathbf{B}$ we write $\mathcal{K}\ell(T)$ for the Kleisli category of T. Its objects are the same as those of $\mathbf{B}$, but its maps $X \to Y$ are the maps $X \to T(Y)$ in $\mathbf{B}$. The unit $\eta\colon X \to T(X)$ is the identity map $X \to X$ in $\mathcal{K}\ell(T)$; and composition of $f\colon X \to Y$ and $g\colon Y \to Z$ in $\mathcal{K}\ell(T)$ is given by $g \odot f = \mu \circ T(g) \circ f$. Maps in such a Kleisli category are understood as computations with outcomes of type T, see [22]. For a monad $T\colon \mathbf{Sets} \to \mathbf{Sets}$ we write $\mathcal{K}\ell_{\mathbb{N}}(T) \hookrightarrow \mathcal{K}\ell(T)$ for the full subcategory with numbers $n \in \mathbb{N}$ as objects, considered as n-element sets.

Proposition 3. *The expectation monad $\mathcal{E}(X) \cong \mathit{Hom}_{PU}(\ell^\infty(X), \mathbb{C})$ gives rise to a full and faithful functor:*

$$\begin{array}{ccc} \mathcal{K}\ell(\mathcal{E}) & \xrightarrow{\quad \mathcal{C}_{\mathcal{E}} \quad} & (\mathbf{CCstar}_{PU})^{op} \\ X & \longmapsto & \ell^\infty(X) \\ \big(X \xrightarrow{f} \mathcal{E}(Y)\big) & \longmapsto & \lambda v \in \ell^\infty(Y).\, \lambda x \in X.\, f(x)(v). \end{array} \tag{4}$$

Proof. First we need to see that $\mathcal{C}_{\mathcal{E}}(f)$ is well-defined: the function $\mathcal{C}_{\mathcal{E}}(f)(v)\colon X \to \mathbb{C}$ must be bounded. We can apply Lemma 1 to the function $f(x) \in \mathit{Hom}_{\mathrm{PU}}(\ell^\infty(Y), \mathbb{C})$;

it yields $\|f(x)(v)\| \leq 4\|v\|$. This holds for each $x \in X$, so that $|\mathcal{C}_{\mathcal{E}}(f)(v)(x)| = |f(x)(v)|$ is bounded by $4\|v\|$. Next, the map $\mathcal{C}_{\mathcal{E}}(f)$ is a PU-map of C^*-algebras via the pointwise definitions of the relevant constructions.

We check that $\mathcal{C}_{\mathcal{E}}$ preserves (Kleisli) identities and composition:

$$\begin{aligned}
\mathcal{C}_{\mathcal{E}}(\mathrm{id})(v)(x) &= \mathcal{C}_{\mathcal{E}}(\eta)(v)(x) \\
&= \eta(x)(v) \\
&= v(x) \\
\mathcal{C}_{\mathcal{E}}(g \odot f)(v)(x) &= (g \odot f)(x)(v) \\
&= \mu\Big(\mathcal{E}(g)(f(x))\Big)(v) \\
&= \mathcal{E}(g)(f(x))\big(\lambda w.\, w(v)\big) \\
&= f(x)\big((\lambda w.\, w(v)) \circ g\big) \\
&= f(x)\big(\lambda y.\, g(y)(v)\big) \\
&= f(x)\big(\mathcal{C}_{\mathcal{E}}(g)(v)\big) \\
&= \mathcal{C}_{\mathcal{E}}(f)\big(\mathcal{C}_{\mathcal{E}}(g)(v)\big)(x) \\
&= \big(\mathcal{C}_{\mathcal{E}}(f) \circ \mathcal{C}_{\mathcal{E}}(g)\big)(v)(x).
\end{aligned}$$

Further, $\mathcal{C}_{\mathcal{E}}$ is obviously faithful, and it is full since for $h\colon \ell^\infty(Y) \to \ell^\infty(X)$ in $\mathbf{CCstar}_{\mathrm{PU}}$ we can define $f\colon X \to \mathit{Hom}_{\mathrm{PU}}(\ell^\infty(Y), \mathbb{C})$ by $f(x)(v) = h(v)(x)$. Then each $f(x)$ is a PU-map of C^*-algebras. □

We turn to the finite case, like in the previous section. We do so by considering the Kleisli category $\mathcal{K}\ell_{\mathbb{N}}(\mathcal{E})$ obtained by restricting to objects $n \in \mathbb{N}$. Since the expectation monad $\mathcal{E}$ and the distribution monad $\mathcal{D}$ coincide on finite sets, we have $\mathcal{K}\ell_{\mathbb{N}}(\mathcal{E}) \cong \mathcal{K}\ell_{\mathbb{N}}(\mathcal{D})$. Maps $n \to m$ in this category are probabilistic transition matrices $n \to \mathcal{D}(m)$. The following equivalence is known, see *e.g.* [20], although possibly not in this categorical form.

Proposition 4. *The functor $\mathcal{C}_{\mathcal{E}}$ from* (4) *restricts in the finite case to an equivalence of categories:*

$$\mathcal{K}\ell_{\mathbb{N}}(\mathcal{D}) \xrightarrow[\simeq]{\mathcal{C}_{\mathcal{D}}} (\boldsymbol{FdCCstar}_{PU})^{op} \tag{5}$$

It is given by $\mathcal{C}_{\mathcal{D}}(n) = \mathbb{C}^n$ and $\mathcal{C}_{\mathcal{D}}\big(n \xrightarrow{f} \mathcal{D}(m)\big) = \lambda v \in \mathbb{C}^m.\, \lambda i \in n.\, \sum_{j \in m} f(i)(j) \cdot v(j)$.

This equivalence (5) may be read as: the category $\mathbf{FdCCstar}_{\mathrm{PU}}$ of finite-dimensional commutative C^*-algebras, with positive unital maps, is the *Lawvere theory* of the distribution monad $\mathcal{D}$.

Proof. Fullness and faithfulness of the functor $\mathcal{C}_{\mathcal{D}}$ follow from Proposition 3, using the isomorphism $\mathit{Hom}_{\mathrm{PU}}(\mathbb{C}^n, \mathbb{C}) \cong \mathcal{D}(n)$ from Lemma 3. This functor $\mathcal{C}_{\mathcal{D}}$ is essentially surjective on objects by Corollary 1, using the fact that a MIU-map is a PU-map. □

5 Continuous Probabilistic Computations

The question arises if the full and faithful functor $\mathcal{K}\ell(\mathcal{E}) \to (\mathbf{CCstar}_{\mathrm{PU}})^{\mathrm{op}}$ from Proposition 3 can be turned into an equivalence of categories, but not just for the finite case like in Proposition 4. In order to make this work we have to lift the expectation monad $\mathcal{E}$ on **Sets** to the category **CH** of compact Hausdorff spaces. As lifting we use what we call the *Radon* monad $\mathcal{R}$, defined on $X \in \mathbf{CH}$ as:

$$\mathcal{R}(X) \;=\; \mathit{Hom}_{\mathrm{PU}}\big(C(X), \mathbb{C}\big), \tag{6}$$

where, as usual, $C(X) = \{f \colon X \to \mathbb{C} \mid f \text{ is continuous}\}$; notice that the functions $f \in C(X)$ are automatically bounded, since X is compact. These are related to measures in the following way. If μ is a probability measure on the Borel sets of X, integration of continuous functions with respect to μ gives $\int_X \text{-}d\mu \in \mathcal{R}(X)$. A Radon probability measure, or an inner regular probability measure, is one such that $\mu(S) = \sup_{K \subseteq S} \mu(K)$ where K ranges over compact sets. The map from measures to elements of $R(X)$ is a bijection[24, Thm. 2.14], and accordingly we shall sometimes refer to elements of $\mathcal{R}(X)$ as measures. Therefore the Radon monad can be considered as a variant of the Giry monad. It differs in that it uses the topology of a space, and that in the case of a non-Polish space there can be non-Radon measures[8, 434K (d), page 192].

This Radon monad $\mathcal{R}$ is known, it first occurs implicitly in [27] as the monad of an adjunction ("probability measure" is used to mean "Radon probability measure" in that article).

From Proposition 1 it is immediate that $\mathcal{R}(X)$ is again a compact Hausdorff space. The unit $\eta \colon X \to \mathcal{R}(X)$ and multiplication $\mu \colon \mathcal{R}^2(X) \to \mathcal{R}(X)$ are defined as for the expectation monad, namely as $\eta(x)(v) = v(x)$ and $\mu(g)(v) = g\big(\lambda h.\, h(v)\big)$. We check that η is continuous. Recall from the proof of Proposition 1 that a basic open in $\mathcal{R}(X)$ is of the form $\mathit{ev}_s^{-1}(U) = \{h \in \mathcal{R}(X) \mid h(s) \in U\}$, where $s \in C(X)$ and $U \subseteq \mathbb{C}$ is open. Then:

$$\eta^{-1}\big(\mathit{ev}_s^{-1}(U)\big) \;=\; \{x \in X \mid \eta(x)(s) \in U\} \;=\; \{x \in X \mid s(x) \in U\} \;=\; s^{-1}(U).$$

The latter is an open subset of X since $s \colon X \to \mathbb{C}$ is a continuous function.

We are now ready to state our main, new duality result. It is may be understood as a probabilistic version of Gelfand duality.

Theorem 2. *The Radon monad* (6) *yields an equivalence of categories:*

$$\mathcal{K}\ell(\mathcal{R}) \;\simeq\; (\mathbf{CCstar}_{PU})^{op}.$$

Proof. We define a functor $\mathcal{C}_{\mathcal{R}} \colon \mathcal{K}\ell(\mathcal{R}) \to (\mathbf{CCstar}_{\mathrm{PU}})^{\mathrm{op}}$ like in (4), namely by:

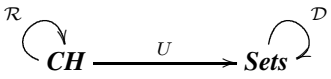
$$\mathcal{C}_{\mathcal{R}}(X) \;=\; C(X) \qquad\qquad \mathcal{C}_{\mathcal{R}}(f) \;=\; \lambda v.\, \lambda x.\, f(x)(v).$$

Since $f \colon X \to \mathcal{R}(Y)$ is itself continuous, so is $f(-)(v) \colon X \to \mathbb{C}$.

The fact that $\mathcal{C}_{\mathcal{R}}$ is a full and faithful functor follows as in the proof of Proposition 3. This functor is essentially surjective on objects by ordinary Gelfand duality (Theorem 1). □

We investigate the Radon monad $\mathcal{R}$ a bit further, in particular its relation to the distribution monad $\mathcal{D}$ on **Sets**.

Lemma 4. *There is a map of monads $(U, \tau)\colon \mathcal{R} \to \mathcal{D}$ in:*



$$\mathcal{D}U \xRightarrow{\tau} U\mathcal{R}$$

where U is the forgetful functor and τ commutes appropriately with the units and multiplications of the monads $\mathcal{D}$ and $\mathcal{R}$. (Such a map is called a "monad functor" in [26, §1].)

As a result the forgetful functor lifts to the associated categories of Eilenberg-Moore algebras:

$$\begin{array}{ccc} \mathcal{EM}(\mathcal{R}) & \longrightarrow & \mathcal{EM}(\mathcal{D}) = \textbf{\textit{Conv}} \\ \big(\mathcal{R}(X) \xrightarrow{\alpha} X\big) & \longmapsto & \big(\mathcal{D}(UX) \xrightarrow{\tau} U\mathcal{R}(X) \xrightarrow{U\alpha} UX\big) \end{array}$$

Hence the carrier of a $\mathcal{R}$-algebra is a convex compact Hausdorff space, and every algebra map is an affine function.

Proof. For $X \in \mathbf{CH}$ and $\varphi \in \mathcal{D}(UX)$, that is for $\varphi\colon UX \to [0,1]$ with finite support and $\sum_x \varphi(x) = 1$, we define $\tau(\varphi) \in U\mathcal{R}(X)$ on $h \in C(X)$ as:

$$\tau(\varphi)(h) \;=\; \textstyle\sum_x \varphi(x) \cdot h(x) \;\in\; \mathbb{C}. \tag{7}$$

It is easy to see that τ is a linear map $C(X) \to \mathbb{C}$ that preserves positive elements and the unit. Moreover, it commutes appropriately with the units and multiplications. For instance:

$$\big(\tau_X \circ \eta^{\mathcal{D}}_{UX}\big)(x)(h) \;=\; \tau_X(1x)(h) \;=\; h(x) \;=\; U(\eta^{\mathcal{R}}_X)(x)(h). \qquad \square$$

The dual space of $C(X)$ can be ordered using (1), by taking the positive cone to be those linear functionals that map positive functions to positive numbers.

Definition 2. *A state $\phi \in \mathcal{R}(X) = \mathit{Hom}_{PU}(C(X), \mathbb{C})$ is a* pure state *if for for each positive linear functional $\psi \leq \phi$ there exists an $\alpha \in [0,1]$ such that $\psi = \alpha\phi$.*

Lemma 5. *For a compact Hausdorff space X, the subset of unit (or Dirac) measures $\{\eta(x) \mid x \in X\} \subseteq \mathcal{R}(X)$ is the set of extreme points of the set of Radon measures $\mathcal{R}(X)$ — where $\eta(x) = \eta^{\mathcal{R}}(x) = \mathsf{ev}_x = \lambda h.\, h(x)$ is the unit of the monad $\mathcal{R}$.*

Proof. We rely on the basic fact, see [5, 2.5.2, page 43], that Dirac measures $\eta(x) \in \mathcal{R}(X)$ are "pure" states. We prove the above lemma by showing that the pure states are precisely the extreme points of the convex set $\mathcal{R}(X)$.

- If $\phi \in \mathcal{R}(X)$ is a pure state, suppose $\phi = \alpha_1\phi_1 + \alpha_2\phi_2$, a convex combination of two states $\phi_i \in \mathcal{R}(X)$ with $\alpha_i \in [0,1]$ satisfying $\alpha_1 + \alpha_2 = 1$, where no two elements of $\{\phi, \phi_1, \phi_2\}$ are the same. Then $\phi \geq \alpha_1\phi_1$, since for a positive function $f \in C(X)$ one has $(\phi - \alpha_1\phi_1)(f) = \alpha_2\phi_2(f) \geq 0$. Thus $\alpha_1\phi_1 = \alpha\phi$, for some $\alpha \in [0,1]$, since ϕ is pure. Then $\alpha_1 = \alpha_1\phi_1(1) = \alpha\phi(1) = \alpha$. If $\alpha_1 = 0$, then $\alpha_2 = 1$ and so $\phi = \phi_2$. If $\alpha_1 > 0$, then $\phi = \phi_1$. Hence ϕ is an extreme point.

– Suppose ϕ is an extreme point of $\mathcal{R}(X)$, *i.e.* that $\phi = \alpha_1\phi_1 + \alpha_2\phi_2$ implies ϕ_1 or $\phi_2 = \phi$. Then if there is a positive linear functional $\psi \leq \phi$, we may take $\alpha_1 = \psi(1) \geq 0$; since $\alpha_1 = \psi(1) \leq \phi(1) = 1$, we get $\alpha_1 \in [0,1]$. If $\alpha_1 = 0$, then since $\|\psi\| = \psi(1) = 0$ we get $\psi = 0$ and $\psi = 0 \cdot \phi$. If $\alpha_1 = 1$, then $(\phi - \psi)(1) = 0$, which since $\phi - \psi$ was assumed to be positive implies $\phi - \psi = 0$ and hence $\psi = 1 \cdot \phi$. Having dealt with those cases, we have that $\alpha_1 \in (0,1)$, and so we have a state $\phi_1 = \frac{1}{\alpha_1}\psi$. We may take $\alpha_2 = 1 - \alpha_1 \in (0,1)$ and obtain a second state $\phi_2 = \frac{1}{\alpha_2}(\phi - \psi)$. By construction we have a convex decomposition of $\phi = \alpha_1\phi_1 + \alpha_2\phi_2$. Therefore either $\phi = \phi_1 = \frac{1}{\alpha_1}\psi$ or $\phi = \phi_2 = \frac{1}{\alpha_2}(\phi - \psi)$. In the first case, $\psi = \alpha_1\phi$, making ϕ pure. But also in the second case ϕ is pure, since we have $\alpha_2\phi = \phi - \psi$ and thus $\psi = (1 - \alpha_2)\phi$. □

Lemma 6. *Let X be a compact Hausdorff space.*

1. *The maps $\tau_X \colon \mathcal{D}(UX) \to U\mathcal{R}(X)$ from* (7) *are injective; as a result, the unit/Dirac maps $\eta \colon X \to \mathcal{R}(X)$ are also injective.*
2. *The maps $\tau_X \colon \mathcal{D}(UX) \rightarrowtail U\mathcal{R}(X)$ are dense.*

Proof. For the first point, assume $\varphi, \psi \in \mathcal{D}(UX)$ satisfying $\tau(\varphi) = \tau(\psi)$. We first show that the finite support sets are equal: $\mathit{supp}(\varphi) = \mathit{supp}(\psi)$. Since X is Hausdorff, singletons are closed, and hence finite subsets too. Suppose $\mathit{supp}(\varphi) \not\subseteq \mathit{supp}(\psi)$, so that $S = \mathit{supp}(\varphi) - \mathit{supp}(\psi)$ is non-empty. Since S and $\mathit{supp}(\psi)$ are disjoint closed subsets, there is by Urysohn's lemma a continuous function $f \colon X \to [0,1]$ with $f(x) = 1$ for $x \in S$ and $f(x) = 0$ for $x \in \mathit{supp}(\psi)$. But then $\tau(\psi)(f) = 0$, whereas $\tau(\varphi)(f) \neq 0$.

Now that we know $\mathit{supp}(\varphi) = \mathit{supp}(\psi)$, assume $\varphi(x) \neq \psi(x)$, for some $x \in \mathit{supp}(\varphi)$. The closed subsets $\{x\}$ and $\mathit{supp}(\varphi) - \{x\}$ are disjoint, so there is, again by Urysohn's lemma a continuous function $f \colon X \to [0,1]$ with $f(x) = 1$ and $f(y) = 0$ for all $y \in \mathit{supp}(\varphi)$. But then $\varphi(x) = \tau(\varphi)(f) = \tau(\psi)(f) = \psi(x)$, contradicting the assumption.

We can conclude that the unit $X \to \mathcal{R}(X)$ is also injective, since its underlying function can be written as composite $U(\eta) = \tau \circ \eta \colon UX \rightarrowtail \mathcal{D}(UX) \rightarrowtail U\mathcal{R}(X)$, because τ is a map of monads.

To show that the image of τ_X is dense, we proceed as follows. By Lemmas 5 and 4, the extreme points of $\mathcal{R}(X)$ are

$$\{\eta^{\mathcal{R}}(x) \mid x \in X\} = \{\tau\big(\eta^{\mathcal{D}}(x)\big) \mid x \in X\}$$

and are thus in the image of $\tau \colon \mathcal{D}(UX) \rightarrowtail U\mathcal{R}(X)$. Since every convex combination of $\eta^{\mathcal{R}}(x)$ comes from a formal convex sum $\varphi \in \mathcal{D}(UX)$, all convex combinations of extreme points are in the image of τ_X. As $\mathcal{R}(X)$ is a compact convex subset of $C(X)^{w*}$ (*i.e.* with the weak-* topology), a locally convex space, we may apply the Krein-Milman theorem [4, Proposition 7.4, page 142] to conclude the set of convex combinations of extreme points is dense. □

Lemma 7. *Let X, Y be compact Hausdorff spaces. Each Eilenberg-Moore algebra $\alpha \colon \mathcal{R}(X) \to X$ is an affine function. For each continuous map $f \colon X \to Y$, the function $\mathcal{R}(f) \colon \mathcal{R}(X) \to \mathcal{R}(Y)$ is affine.*

Proof. This follows from naturality of $\tau\colon \mathcal{D}U \Rightarrow U\mathcal{R}$. □

Proposition 5. *Let $\alpha\colon \mathcal{R}(X) \to X$ and $\beta\colon \mathcal{R}(Y) \to Y$ be two Eilenberg-Moore algebras of the Radon monad $\mathcal{R}$. A function $f\colon X \to Y$ is an algebra homomorphism if and only f is both continuous and affine.*

As a result, the functor $\mathcal{EM}(\mathcal{R}) \to \mathcal{EM}(\mathcal{D}) = \mathbf{Conv}$ from Lemma 4 is full and faithful.

We shall follow the convention of writing $\mathcal{A}(X, Y)$ for the homset of continuous and affine functions $X \to Y$.

Proof. Clearly, each algebra map is both continuous and affine. For the converse, if $f\colon X \to Y$ is continuous, it is a map in the category **CH** of compact Hausdorff spaces. Since it is affine, both triangles commute in:

$$\begin{array}{ccc} \mathcal{D}(UX) & \xrightarrow[\text{dense}]{\tau} & \mathcal{R}(X) \\ & \searrow^{f\circ\alpha} \quad {\scriptstyle \beta\circ\mathcal{R}(f)}\downarrow & \\ & Y & \end{array}$$

Since Y is Hausdorff, there is at most one such map. □

The category $\mathcal{EM}(\mathcal{R})$ of Eilenberg-Moore algebras of the Radon monad may thus be understood as a suitable category of convex compact Hausdorff spaces, with affine continuous maps between them. We side-step its precise characterisation — in order to avoid "observability" issues like in [16] — and will proceed with $\mathcal{EM}(\mathcal{R})$ as such. Details will be elaborated in an extended version of this paper.

6 States and Effects

We start with a simple observation.

Lemma 8. *The unit interval $[0, 1]$ is obviously compact and Hausdorff. It carries a $\mathcal{R}$-algebra structure $\mathcal{R}([0, 1]) \to [0, 1]$, given by $h \mapsto h\big([0, 1] \hookrightarrow \mathbb{C}\big)$.*

For an arbitrary $\mathcal{R}$-algebra X, the homset of algebra maps:

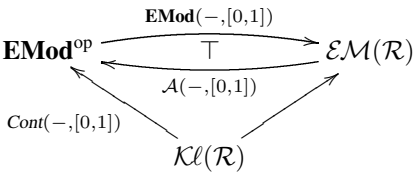
$$\mathcal{EM}(\mathcal{R})\big(X, [0, 1]\big) = \mathcal{A}(X, [0, 1])$$

is an effect module, via pointwise constructions. Recall from Proposition 5 that this homset contains the affine and continuous functions $X \to [0, 1]$. In this way we get a functor $\mathcal{A}(-, [0, 1])\colon \mathcal{EM}(\mathcal{R}) \to \mathbf{EMod}^{op}$. □

In [16] it is shown that for an effect module M, the homset $\mathbf{EMod}(M, [0, 1])$ is a convex compact Hausdorff space. In fact, it carries an $\mathcal{R}$-algebra structure:

$$\begin{array}{ccc} \mathcal{R}\big(\mathbf{EMod}(M, [0, 1])\big) & \xrightarrow{\alpha_M} & \mathbf{EMod}(M, [0, 1]) \\ h & \longmapsto & \lambda x \in M.\, h(\mathrm{ev}_x) \end{array}$$

where $\mathit{ev}_x = \lambda v.\, v(x) \colon C\big(\mathbf{EMod}(M,[0,1])\big) \to \mathbb{C}$. For each map of effect modules $f \colon M \to M'$ one obtains a map of $\mathcal{R}$-algebras $(-) \circ f \colon \mathbf{EMod}(M',[0,1]) \to \mathbf{EMod}(M,[0,1])$. We thus obtain the following situation:



(8)

Such diagrams appear in [13] as a categorical representation of the duality between states and effects, with Schrödinger's picture on the right, and Heisenberg's picture on the left (see also [15]). In this diagram:

- The map $\mathcal{K}\ell(\mathcal{R}) \to \mathbf{EMod}^{\mathrm{op}}$ on the left is the "predicate" functor, sending a space X to the predicates on X, given by the effect module $\mathit{Cont}(X,[0,1])$ of continuous functions $X \to [0,1]$, or equivalently by the effects $[0,1]_{C(X)}$ on the C^*-algebra $C(X)$. This functor is full and faithful by Lemma 2 and Theorem 2:

$$\begin{aligned}\mathbf{EMod}\big(\mathit{Cont}(Y,[0,1]),\mathit{Cont}(X,[0,1])\big) &= \mathbf{EMod}\big([0,1]_{C(Y)},[0,1]_{C(X)}\big)\\ &\cong \mathit{Hom}_{\mathrm{PU}}\big(C(Y),C(X)\big)\\ &\cong \mathcal{K}\ell(\mathcal{R})\big(X,Y\big).\end{aligned}$$

- The "state" functor $\mathcal{K}\ell(\mathcal{R}) \to \mathcal{EM}(\mathcal{R})$ is the standard full and faithful "comparison" functor from a Kleisli category to a category of Eilenberg-Moore algebras.
- The diagram (8) commutes in one direction:

$$\begin{aligned}\mathbf{EMod}\big(\mathit{Cont}(X,[0,1]),[0,1]\big) &= \mathbf{EMod}\big([0,1]_{C(X)},[0,1]_{\mathbb{C}}\big)\\ &\cong \mathit{Hom}_{\mathrm{PU}}\big(C(X),\mathbb{C}\big) = \mathcal{R}(X).\end{aligned}$$

- The remainder of this section will be devoted to proving that the diagram also commutes in the other direction, *i.e.* $\mathcal{A}(\mathcal{R}(X),[0,1]) \cong \mathit{Cont}(X,[0,1])$.

There is an evaluation map ζ from $C(X)$ to $\mathcal{R}(X) \to \mathbb{C}$ defined as follows:

$$\zeta(f) = \mathit{ev}_f = \lambda\phi.\, \phi(f).$$

Lemma 9. *For $f \in C(X)$, this $\zeta(f)$ is affine and continuous, so $\zeta(f) \in \mathcal{A}(\mathcal{R}(X),\mathbb{C})$.*

Proof. The map $\zeta(f)$ is continuous because for an open $U \subseteq \mathbb{C}$ the inverse image $\zeta(f)^{-1}(U) = \mathit{ev}_f^{-1}(U)$ is by definition a basic open of the weak-* topology on $\mathcal{R}(X)$. It is also affine, since it sends convex sums in $\mathcal{R}(X)$ to convex sums in $\mathbb{C}$:

$$\zeta(f)\big(\textstyle\sum_i r_i\phi_i\big) = \zeta(f)\big(\lambda g.\, \sum_i r_i\phi_i(g)\big) = \sum_i r_i\phi_i(f) = \sum_i r_i\zeta(f)(\phi_i). \quad \square$$

The following lemma is a special case of the complexification of [2, proposition 2.2]. However, a simpler proof is possible in this special case by applying Gelfand duality, which we include here.

Lemma 10. *The set $\mathcal{A}(\mathcal{R}(X), \mathbb{C})$ can be given (pointwise) the structure of an ordered vector space with unit.*

Moreover, this space is isomorphic to $C(X)$, with one direction given by ζ.

Proof. The complex numbers are an ordered vector space with positive cone $[0, \infty) \subseteq \mathbb{C}$ and unit $1 \in \mathbb{C}$. By treating $U(\mathcal{R}(X))$ as a set, we obtain an ordered unital vector space structure on $\mathbb{C}^{U(\mathcal{R}(X))}$ as an infinite product. Therefore we need only show that the affine and continuous maps are closed under linear combinations. The latter holds because addition and scalar multiplication are continuous on $\mathbb{C}$, so we reduce to the former. If we consider a finite linear combination of maps $f = \sum_i \alpha_i f_i$ with each $f_i \in \mathcal{A}(\mathcal{R}(X), \mathbb{C})$, we may use the fact that all summations involved are finite and the associativity of addition to get that f is affine from the fact that each f_i is.

We proceed to the second part of the statement. By lemma 9 we have that ζ maps into $\mathcal{A}(\mathcal{R}(X), \mathbb{C})$. We need to show that it is a linear map preserving the positive cone and unit. To do this we use the fact that ζ is an evaluation map and that each ϕ is linear and preserves the positive cone and unit, being a state.

At this point we note that if $\zeta(f)(\phi) \in [0, \infty)$ for all $\phi \in \mathcal{R}(X)$ then f is positive. To see this, assume f has this property. Then in particular $\zeta(f)(\phi) \geq 0$ for all pure states ϕ in the spectrum of $C(X)$. Since ζ is evalution, $f(x) \geq 0$ at each point.

This shows that if ζ has an inverse, it is a positive map. We also have that if ζ has an inverse, it would preserve the unit, so to show that ζ is an isomorphism of ordered vector spaces with unit we only have to show that it is a bijection.

To show ζ is injective, assume that there are $f, g \in C(X)$ such that $\zeta(f) = \zeta(g)$. Then $\zeta(f)$ agrees with $\zeta(g)$ at each pure state in the spectrum of $C(X)$, *i.e.* f agrees with g at each point $x \in X$, and so $f = g$.

To show ζ is surjective, let $f \in \mathcal{A}(\mathcal{R}(X), \mathbb{C})$. By restriction we obtain a continuous function $\mathrm{Spec}(C(X)) \hookrightarrow \mathcal{R}(X) \to \mathbb{C}$; it corresponds to a unique element $g \in C(X)$, *i.e.* to a map $g \colon X \to \mathbb{C}$, by Gelfand's isomorphism. We need to show that $\zeta(g) = f$. We know $\zeta(g)$ agrees with f on the spectrum, and by affineness they must agree on all convex combinations of these. Using Lemma 6 (2) we see that they agree on a dense set, so $\zeta(g) = f$ by continuity. □

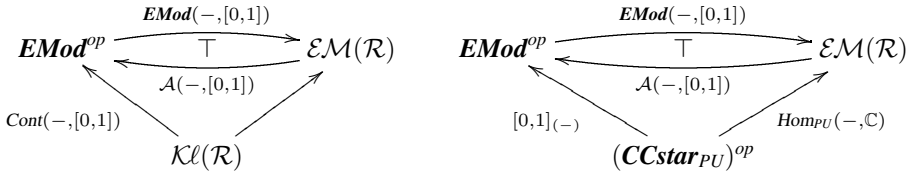
Corollary 3. *For each commutative C^*-algebra A there is an isomorphism of ordered vector spaces with unit: $A \cong \mathcal{A}\big(\mathbf{Hom}_{PU}(A, \mathbb{C}), \mathbb{C}\big)$.* □

Proof. Using Gelfand duality, we extend the above to all commutative C^*-algebras, via $A \cong C(\mathrm{Spec}(A))$. □

Proposition 6. *The $\mathcal{R}$-algebra maps $\mathcal{A}(\mathcal{R}(X), [0,1])$, with their pointwise effect module structure, are isomorphic to predicates $\mathbf{Cont}(X, [0,1])$ on X, i.e. to the effects $[0,1]_{C(X)}$ on $C(X)$.*

Proof. By Lemma 10, there is an isomorphism $C(X) \cong \mathcal{A}(\mathcal{R}(X)), \mathbb{C})$ of unital ordered vector spaces. Restriction to their intervals from zero to the unit then yields an isomorphism $[0,1]_{C(X)} \cong \mathcal{A}(\mathcal{R}(X)), [0,1])$. Recalling Proposition 5, we see the latter maps are exactly the $\mathcal{R}$-algebra maps. □

Theorem 3. *There are commuting "state-and-effect" triangles:*



Proof. The triangle on the left is the diagram (8), in which the missing commutation result is given by Proposition 6. The diagram on the right follows from the equivalence $\mathcal{K}\ell(\mathcal{R}) \simeq (\mathbf{CCstar}_{\mathrm{PU}})^{\mathrm{op}}$ from Theorem 2. □

Final Remarks

The main contribution of this article lies in establishing a connection between two different worlds, namely the world of theoretical computer scientists using program language semantics (and logic) via monads, and the world of mathematicians and theoretical physicists using C^*-algebras. This connection involves the distribution monad $\mathcal{D}$ on **Sets**, which is heavily used for modeling discrete probabilistic systems (Markov chains), in the finite-dimensional case (see Proposition 4) and the less familiar Radon monad $\mathcal{R}$ on compact Hausdorff spaces (see Theorem 2). These results apply to *commutative* C^*-algebras. Follow-up research will concentrate on the non-commutative case.

Acknowledgements. The authors wish to thank Hans Maassen and Jorik Mandemaker for helpful discussions.

References

1. Abramsky, S., Coecke, B.: A categorical semantics of quantum protocols. In: Engesser, K., Gabbai, D.M., Lehmann, D. (eds.) Handbook of Quantum Logic and Quantum Structures, pp. 261–323. North Holland, Elsevier, Computer Science Press (2009)
2. Alfsen, E.M., Shultz, F.W.: State Spaces of Operator Algebras. Birkhaäuser (2001)
3. Arveson, W.: An Invitation to C^*-Algebra. Springer (1981)
4. Conway, J.B.: A Course In Functional Analysis, 2nd edn. Graduate Texts in Mathematics, vol. 96. Springer (1990)
5. Dixmier, J.: C^*-Algebras. North-Holland Mathematical Library, vol. 15. North-Holland Publishing Company (1977)
6. Dvurečenskij, A., Pulmannová, S.: New Trends in Quantum Structures. Kluwer Acad. Publ., Dordrecht (2000)
7. Foulis, D.J., Bennett, M.K.: Effect algebras and unsharp quantum logics. Found. Physics 24(10), 1331–1352 (1994)
8. Fremlin, D.H.: Measure Theory, vol. 4 (2003), `http://www.essex.ac.uk/maths/people/fremlin/mt.htm`
9. Girard, J.-Y.: Geometry of Interaction V: Logic in the hyperfinite factor. Theor. Comput. Sci. 412(20), 1860–1883 (2011)

10. Giry, M.: A categorical approach to probability theory. In: Banaschewski, B. (ed.) Categorical Aspects of Topology and Analysis. Lecture Notes in Mathematics, vol. 915, pp. 68–85. Springer, Heidelberg (1982)
11. Heinosaari, T., Ziman, M.: The Mathematical Language of Quantum Theory. From Uncertainty to Entanglement. Cambridge Univ. Press (2012)
12. Horodecki, M., Horodecki, P., Horodecki, R.: Separability of Mixed States: Necessary and Sufficient Conditions. Physics Letters A 223(12), 1–8 (1996)
13. Jacobs, B.: Introduction to Coalgebra. Towards Mathematics of States and Observations, Book, in preparation; version 2.0 (2012), `http://www.cs.ru.nl/B.Jacobs/CLG/JacobsCoalgebraIntro.pdf`
14. Jacobs, B.: Involutive categories and monoids, with a GNS-correspondence. Found. of Physics 42(7), 874–895 (2012)
15. Jacobs, B.: Measurable spaces and their effect logic. In: Logic in Computer Science. IEEE Computer Science Press (2013)
16. Jacobs, B., Mandemaker, J.: The expectation monad in quantum foundations. In: Jacobs, B., Selinger, P., Spitters, B. (eds.) Quantum Physics and Logic (QPL) 2011. Elect. Proc. in Theor. Comp. Sci., vol. 95, pp. 143–182 (2012)
17. Jacobs, B., Mandemaker, J.: Relating operator spaces via adjunctions. In: Chubb Reimann, J., Harizanov, V., Eskandarian, A. (eds.) Logic and Algebraic Structures in Quantum Computing and Information. Lect. Notes in Logic, Cambridge Univ. Press (2013), See arxiv.org/abs/1201.1272
18. Johnstone, P.: Stone Spaces. Studies in Advanced Mathematics, vol. 3. Cambridge Univ. Press (1982)
19. Kock, A.: On double dualization monads. Math. Scand. 27, 151–165 (1970)
20. Maassen, H.: Quantum probability and quantum information theory. In: Benatti, F., Fannes, M., Floreanini, R., Petritis, D. (eds.) Quantum Information, Computation and Cryptography. Lect. Notes Physics, vol. 808, pp. 65–108. Springer, Berlin (2010)
21. Mislove, M., Ouaknine, J., Pavlovic, D., Worrell, J.: Duality for labelled Markov processes. In: Walukiewicz, I. (ed.) FOSSACS 2004. LNCS, vol. 2987, pp. 393–407. Springer, Heidelberg (2004)
22. Moggi, E.: Notions of computation and monads. Inf. & Comp. 93(1), 55–92 (1991)
23. Wick Pelletier, J., Rosický, J.: On the Equational Theory of C^*-algebras. Algebra Universalis 30, 275–284 (1993)
24. Rudin, W.: Functional Analysis, 3rd edn. McGraw-Hill Book Company (1987)
25. Sakai, S.: C^*-algebras and W^*-algebras. Ergebnisse der Mathematik und ihrer Grenzgebiete, vol. 60. Springer (1971)
26. Street, R.: The formal theory of monads. Journ. of Pure & Appl. Algebra 2, 149–169 (1972)
27. Swirszcz, T.: Monadic Functors and Convexity. Bulletin de l'Académie Polonaise des Sciences, Série des Sciences Math. Astr. et Phys. 22(1), 39–42 (1974)

Trace Semantics via Generic Observations

Sergey Goncharov

Department of Computer Science,
Friedrich-Alexander-Universität Erlangen-Nürnberg

Abstract. Recent progress on defining abstract trace semantics for coalgebras rests upon two observations: (i) coalgebraic bisimulation for deterministic automata coincides with trace equivalence, and (ii) the classical powerset construction for automata determinization instantiates the generic idea of lifting a functor to the Eilenberg-Moore category of an appropriate monad $\mathbb{T}$. We take this approach one step further by rebasing the latter kind of trace semantics on the novel notion of $\mathbb{T}$-*observer*, which is just a certain natural transformation of the form $F \to GT$, and thus allowing for elimination of assumptions about the structure of the coalgebra functor. As a specific application of this idea we demonstrate how it can be used for capturing trace semantics of push-down automata. Furthermore, we show how specific forms of observers can be used for coalgebra-based treatment of internal automata transitions as well as weak bisimilarity of processes.

1 Introduction

Perhaps the most impressive and productive category-theoretic archetypes adapted by theoretical computer science are the notions of coalgebra and computational monad. Whereas computational monads are typically used for sakes of denotational semantics in order to encapsulate, i.e. internalize a computational effect, and thus make it invisible, coalgebras are better known for their extroverted character, exhibited by their tendency to actively interact with the outside.

As is usually the case, except oversimplified, one needs both kinds of features: a way to hide some information, but also a possibility to stay reactive. Generic trace semantics for coalgebras, originated in [11], can be viewed as an attempt to resolve the mismatch between internal and external behaviours of coalgebras by means of the generic notion of a trace, provided one treats nondeterminism as a sort of intrinsic effect to be abstracted away from. More recently in [13], it has been ascertained that generic traces naturally appear in a generalization of the powerset construction for nondeterministic automata by identifying the powerset functor as a monad and abstracting away from it. In view of the latter work, the distinction between coalgebraic behaviours and traces can be explained most easily by the core example of nondeterministic automata as follows. A nondeterministic automaton is presented by a coalgebra of the functor $F\mathcal{P}_\omega$ where $\mathcal{P}_\omega$ stands for finite powersets and F is the deterministic automata functor $2 \times -^A$. The universal arrow from such a coalgebra to the final coalgebra $\nu F\mathcal{P}_\omega$ captures behaviours, while traces are obtained as behaviours of the determinized

R. Heckel and S. Milius (Eds.): CALCO 2013, LNCS 8089, pp. 158–174, 2013.

version of the original automaton. The underlying property involved here is a fact, known from experience: for deterministic automata, trace equivalence and behavioural equivalence coincide. Notably, the theory does not explain why this is the case. Moreover, the obtained trace semantic becomes rigidly bound to the syntactic form of the functor, which limits the application power of the construction drastically.

In this work, which is heavily based on [13] as well as on the more recent [28], we attempt to push the existing development forward in two respects. First, we generalise the framework by introducing the concept of a $\mathbb{T}$-*observer*, which is a natural transformation of the form

$$F \to GT$$

with G subject to a distributive law $\pi : \mathbb{T}G \to G\mathbb{T}$. This will allow us to detach from any assumptions about the shape of the coalgebra functor; Therefore, we shall see how this helps to naturally define trace semantics of push-down automata (and thus refine the earlier attempt from [28]). Second, we establish the notion of $\mathbb{T}$-observer as a generic phenomenon, arising from a natural adjoint construction. The latter, somewhat surprisingly, suggests a distinction between *real-time observers*, viewing only the current transition of the system, and *lookahead observers*, viewing potentially infinite sequence of transitions starting from the current one on.

Paper Organization. In Sections 2 and 3 we give the necessary preliminaries on coalgebras and (computational) monads; Moreover, in the latter we introduce a stack monad and its nondeterministic counterpart and give operational characterizations for them. In Section 4 we introduce the crucial notion of a $\mathbb{T}$-observer and then, in Section 5, reesteblish it by a category-theoretic argument. Sections 6, 7 and 8 contain the main corpus of examples: internal actions of automata, weak bisimilarity, pushdown automata, and reactive programs with side-effects.

2 Systems and (Final) Coalgebras

In general, one can sensibly speak about a system as of something sequentially evolving in time while changing its internal state and possibly interacting with the outer world. The notion of *coalgebra* is known to provide a suitable mathematical abstraction, basically, as general as that.

Given a category $\mathbf{C}$ (typically the category **Set** of sets and functions) and an endofunctor $F : \mathbf{C} \to \mathbf{C}$, an F-coalgebra is any morphism of the form $f : X \to FX$. The object X it often referred to as the *state space* of f. For a fixed F, F-coalgebras form a category, which we denote as $\mathsf{coalg}_F(\mathbf{C})$, with morphisms being the morphisms of the state spaces, subject to coherence with the coalgebra structure as follows: for $f : X \to FX$ and $g : Y \to FY$, $h : X \to Y$ is an F-morphism iff $gh = (Fh)f$. A terminal object in the category of F-coalgebras, called a *final coalgebra*, (whose state space is) often denoted as νF, if it exists, plays a critical role in the theory of systems: given an F-coalgebra $f : X \to FX$, the terminal morphism $\hat{f} : X \to \nu F$ produces *behaviours* of the system,

i.e. the complete characterization of its evolution for every chosen initial state $s : 1 \to X$. States with the same behaviour are called *behaviourally equivalent*. For weak pullback preserving functors on **Set**, behavioural equivalence is known to capture the core notion of *strong bisimilarity*, stemming from the realm of process algebra [26]

We now recap two quite representative examples (see e.g. [13]): deterministic (DA) and nondeterministic (NA) automata.

Example 1 (DA). Let $FX = 2 \times X^A$ be an endofunctor over **Set** with finite A, understood as an alphabet of atomic symbols. A coalgebra of this functor has the form $\langle o, t\rangle : X \to 2 \times X^A$. With X being finite $\langle o, t\rangle$ represents a deterministic automaton: $o : X \to 2$ models the acceptance condition, equivalently a subset of final states; and $t : X \to X^A$, whose uncurried version has the profile $X \times A \to X$, models the transition function.

The final coalgebra has as the state space the set of all formal languages over A, i.e. $\nu F = 2^{A^*}$. This can be equipped with the F-coalgebra structure by the isomorphism:

$$\iota : 2^{A^*} \simeq 2^{1+A\times A^*} \simeq 2 \times (2^{A^*})^A.$$

Now the final map $\hat{f}$ generated by $f = \langle o, t\rangle$ sends every state x of the automaton to the language accepted by f at this state.

The functor $2 \times -^A$ plays a particular role in trace semantics. We denote it by L_A and call *(formal) language functor*, which name is suggested by the form of the final coalgebra νL_A. We shall, when appropriate, treat the elements $0, 1$ of 2 as truth values $\bot$ and $\top$ correspondingly.

Example 2 (NA). In order to switch to the nondeterministic case it suffices to take the functor by $FX = 2 \times \mathcal{P}_\omega(X)^A$. The transition function of an NA $\langle c, t\rangle : X \to 2 \times \mathcal{P}_\omega(X)^A$ now becomes equivalent to a finitary relation over $X \times A \times X$. The state space of the final F-coalgebra consists of all finitely-branching trees whose nodes are labelled either by $\bot$ or by $\top$ and whose edges are labelled by action names from A, modulo a suitable bisimilarity relation.

As indicated above, although DA and NA recognize the same languages, coalgebra-driven behavioural equivalences diverge for them.

3 Monads, Theories, and Effects

Monads play a crucial role in denotational semantics, for they capture the very essence of various side-effects, most prominently their composability [19, 18]. Monads are also known to be successfully applied in coalgebra-based formalisations of systems, although in a somewhat restricted way. The underlying philosophy of this paper is to identify monads with computational effects in spirit of [19]. In accordance to this we view e.g. nondeterministic automata as those featuring nondeterminism as a side-effect, etc.

Given a Cartesian category **C**, i.e. a category with finite Cartesian products (including the empty one, which is the terminal object), a monad $\mathbb{T}$ can be given by a so-called *Kleisli triple* $(T, \eta, -^\dagger)$ consisting of an endomap $T : \mathrm{Ob}(\mathbf{C}) \to \mathrm{Ob}(\mathbf{C})$;

a family of morphisms $\eta_A : A \to TA$; and an operator sending any $f : A \to TB$ to $f^\dagger : TA \to TB$, called *Kleisli lifting*. These data are subject to the equations

$$\eta_A^\dagger = \mathrm{id}, \qquad f^\dagger \eta_A = f, \qquad (f^\dagger g)^\dagger = f^\dagger g^\dagger.$$

Intuitively, T is used to form a type of computations TA with outcomes of type A; Therefore η_A injects a value into a trivial computation returning that value and $-^\dagger$ lifts a morphism $f : A \to TB$ over values to a morphism $f^\dagger : TA \to TB$ over computations.

It can be shown that T from the definition of the Kleisli triple can be lifted to an endofunctor and η_A is natural in A, which leads us to an equivalent definition of a monad, customary in the category theory: A monad $\mathbb{T}$ is given by an endofunctor T and two natural transformations $\eta : Id \to T$ and $\mu : T^2 \to T$ called *unit* and *multiplication* respectively, subject to standard commutativity diagrams (see e.g. [16]). A monad $\mathbb{T}$ is *strong* if it has *strength*, a natural transformation $\tau_{A,B} : A \times TB \to T(A \times B)$, which is well-behaved w.r.t. the monad structure [19]. Strength is a natural technical condition ensuring that the monad is suitable for multivariable computations. If strength exists then it is unique up to natural isomorphism. Moreover, for Cartesian closed categories strength is equivalent to enrichment [15]; Therefore, every monad over **Set** is strong. We agree that all the monads in remainder of this paper are strong.

The definition of a monad by a Kleisli triple naturally suggests the idea of the *Kleisli category* $\mathbf{C}_\mathbb{T}$, the category whose objects are the same as those of $\mathbf{C}$ and whose morphisms from $\mathbf{C}_\mathbb{T}(A, B)$ are those from $\mathbf{C}(A, TB)$; The identity morphisms of $\mathbf{C}_\mathbb{T}$ are thus given by $\eta_A : A \to TA$ and the composition of $f : B \to TC$ with $g : A \to TB$ is the so-called *Kleisli composition*: $f \diamond g = f^\dagger g$. Morphisms of $\mathbf{C}_\mathbb{T}$ are sometimes called *Kleisli morphisms*. In terms of computational effects, the Kleisli category is precisely the category of side-effecting morphisms w.r.t. $\mathbf{C}$. In particular, one can "include" $\mathbf{C}$ in $\mathbf{C}_\mathbb{T}$ by postcomposing every morphism $f : A \to B$ with η_B. The obtained functor has a right adjoint and as such gives rise to a monad, which happens to be the original monad $\mathbb{T}$. This provides one of the two extremal solutions to the question, if any monad is generated by an adjunction. The other solution, known as *Eilenberg-Moore construction*, is obtained by involving the so-called *Eilenberg-Moore category* $\mathbf{C}^\mathbb{T}$, i.e. the category of algebras of the monad $\mathbb{T}$. Such algebras are simply morphisms $f : TA \to A$ satisfying certain coherence conditions [16]. The Kleisli category $\mathbf{C}_\mathbb{T}$ can be faithfully embedded into $\mathbf{C}^\mathbb{T}$ as the subcategory of free algebras, which gives rise to a functor $\mathcal{F}^\mathbb{T}_{\mathrm{EM}} : \mathbf{C} \to \mathbf{C}^\mathbb{T}$ that has a right adjoint $\mathcal{G}^\mathbb{T}_{\mathrm{EM}} : \mathbf{C}^\mathbb{T} \to \mathbf{C}$ and therewith again yields the original monad $\mathbb{T}$.

Plain dualization of the monadic universe brings about *comonads* and a whole bunch of associated concepts, which we do not elaborate here (but see [30]). We note, however, that the Eilenberg-Moore category $\mathsf{Coalg}_\mathbb{K}(\mathbf{C})$ of a comonad $\mathbb{K}$ consists of Eilenberg-Moore $\mathbb{K}$-coalgebras. The latter category is not the same as $\mathsf{coalg}_K(\mathbf{C})$ with K being the functorial part of $\mathbb{K}$, but closely related to it. In particular, $\mathsf{Coalg}_\mathbb{K}(\mathbf{C})$ is a full subcategory of $\mathsf{coalg}_K(\mathbf{C})$.

Examples of computationally relevant monads include the following.

Example 3 (Computational monads). We assume that the category $\mathbf{C}$ possesses sufficient structure to makes sense of what follows.

- *Exception monad:* $TX = X + E$ where E is an object of exceptions. One obtains the *partiality monad* by taking $E = 1$.
- *Powerset monad:* $TX = \mathcal{P}X$ where $\mathcal{P}$ is a covariant powerset functor. Some variants of it are $\mathcal{P}^\star$ for non-empty subsets; $\mathcal{P}_\kappa$ for subsets of cardinality strictly less then κ with a regular cardinal κ (e.g. finite powerset $\mathcal{P}_\omega$); etc.
- *Multiset monad:* $TX = \{m : X \to \mathbb{N} \mid |\mathrm{supp}(m)| < \omega\}$ where $\mathbb{N}$ stands for the set of natural numbers including 0.
- *Subdistribution monad:* $TA = \left\{ d : A \to [0,1] \mid \sum_{x \in A} d(x) \leq 1 \right\}$ where d ranges over subprobability distributions, deviating from probability distributions in that they might sum up to less than 1.
- *Store monad:*[1] $TX = (X \times S)^S$ where S is a global store often identified with V^L, a space of maps from locations L to values V.

Often, different effects can be combined, e.g. the nondeterministic store monad $TX = \mathcal{P}(X \times S)^S$, the Java monad [12] $TA = S \rightharpoonup S \times A + E \times A$, etc.

For certain monads, especially those involved in coalgebraic trace semantics, it is customary to consider their presentation, based on algebraic theories. For example, the finite powerset monad $\mathcal{P}_\omega$ can be considered as generated by the algebraic theory featuring two operations $\emptyset$ and $+$, subject to the axioms of bounded semi-latices (i.e. semi-lattices with a bottom element). The object $\mathcal{P}_\omega X$ is then identified with the free bounded semi-lattice over X. More recently, it has been shown how the store monad can be presented by an algebraic theory, which example we consider in more detail.

Consider the store S of the form V^n with natural n. This corresponds to a computational model of n locations that can be filled with the elements of V. Let the underlying category be **Set** and assume, for simplicity, V to be finite. We consider the family of operations:

$$lookup_i : X^V \to X \qquad\qquad update_{i,v} : X \to X$$

parametrized by $i \in n$ and $v \in V$. The intuitive meaning of them is as follows: $lookup_i(x_1, \ldots, x_V)$ reads the location i and depending on the value discovered, returns the corresponding argument; $update_{i,v}(x)$ updates the location i with v and returns x. These two operations can be viewed as elementary commands, while the terms composed from them can be viewed as programs over these commands. A complete set of axioms for *lookup* and *update* is provided in [24]; Moreover the following result is proven.

Proposition 4. [24] *The axioms for lookup and update from [24] identify an equational theory generating the store monad over* V^n.

The approach to computational effects sketched above is developed mainly in a series of work by Plotkin and Power [21, 22, 23] and scales rather well as far as to any strong monad over a complete and cocomplete Cartesian closed **C** [5].

[1] We avoid the term "state monad" to prevent confusion with coalgebraic states.

Various forms of memory organization naturally call for various algebraic theories. We conclude this section by giving two novel characterisation results in spirit of Proposition 4 for stack-like stores. Analogously to *lookup* and *update* we introduce $pop : X^{S+1} \to X$ and $push_i : X \to X$, with $S = \{s_1, \ldots, s_n\}$ denoting a finite alphabet of stack elements. The idea is as follows:

- $pop(x_1, \ldots, x_n, y)$ results in y if the stack is empty; Otherwise it removes the top element of it and results in x_i where $s_i \in S$ is the removed stack element.
- $push_i(x)$ pushes $s_i \in S$ onto the stack and returns x.

We postulate the following axioms:

$$push_i(pop(x_1, \ldots, x_n, y)) = x_i \tag{1}$$
$$pop(push_1(x), \ldots, push_n(x), x) = x \tag{2}$$
$$pop(x_1, \ldots, x_n, pop(y_1, \ldots, y_n, z)) = pop(x_1, \ldots, x_n, z) \tag{3}$$

The *stack monad* is the submonad of the store monad $(- \times S^*)^{S^*}$ with every TX consisting of those $\langle r, t\rangle : S^* \to X \times S^*$ for which there is a natural $n \geq 0$ such that whenever $w, u \in S^*$ with $|u| > n$,

$$t(u \cdot w) = t(u) \cdot w \qquad\qquad r(u \cdot w) = t(u)$$

In other words, TX captures exactly those operations $\langle r, t\rangle$, which may only access the stack up to a certain depth n where n is associated with $\langle r, t\rangle$ and independent from the stack content.

Proposition 5. *The algebraic theory of $push_i$ and pop with the axioms* (1)–(3) *is equivalent to the stack monad over* **Set**.

A more advanced theory accommodating stacks together with finite nondeterminism will be needed in Section 7 in order to give a coalgebraic account of push-down automata. It is obtained by adding binary $+$ and nullary $\varnothing$, to the signature of operations $\{push_i \mid i \leq n\} \cup \{pop\}$ used above and by completing the axioms (1)–(3) with the following new identities:

$$(x + y) + z = x + (y + z) \qquad x + y = y + x \qquad x + \varnothing = x + x = x \tag{4}$$

$$pop(\varnothing) = \varnothing \tag{5}$$
$$push(\varnothing, \ldots, \varnothing, \varnothing) = \varnothing \tag{6}$$
$$pop(x + x') = pop(x) + pop(x') \tag{7}$$
$$push(x_1 + x'_1, \ldots, x_n + x'_n, y + y') = push(x_1, \ldots, x_n, y) + push(x'_1, \ldots, x'_n, y') \tag{8}$$

Here (4) are the obvious axioms of bounded semi-lattices whereas the laws (5)–(8) express commutativity of stack operations and nondeterminism over each other as computational effects. In other words, the theory for (1)–(8) is the tensor product of the theory for stacks (1)–(3) and the theory for finite nondeterminism (4) (see [10] for more details). The corresponding *nondeterministic*

stack monad is the submonad of the nondeterministic store monad $\mathcal{P}_\omega(- \times S^*)^{S^*}$ so that every TX consists of those $f : S^* \to \mathcal{P}_\omega(X \times S^*)$ for which there is a natural $n \geq 0$ such that for all $w, u \in S^*$ whenever $|u| > n$,

$$f(u \cdot w) = \{\langle x, v \cdot w\rangle \mid \langle x, v\rangle \in f(u)\}.$$

Proposition 6. *The algebraic theory of push, pop, $\varnothing$ and $+$ with the axioms* (1)–(8) *is equivalent to the nondeterministic stack monad over* **Set**.

The latter proposition is essentially a consequence of the analysis of the structure of *powermonads* given in [8], which are tensor products of monads with variants of the powerset monad.

4 Trace Semantics via Observation

We consider here a monad $\mathbb{T}$ and a pair of endofunctors F, G over a category $\mathbf{C}$ having sufficient structure to interpret the constructions being discussed.

As indicated previously, process-like bisimilarity is rather well captured by coalgebraic behavioural equivalence. Proper treatment of *trace equivalence*, however, turns out to be a rather more delicate issue and a seemingly prevailing approach to tackle it, originated in [11], is to assume F to be a composite functor and capture trace equivalence using the same finality argument in another category with a functor obtained as a syntactic component of F. One concrete implementation of this idea is to assume F to be of the form TG and obtain the trace semantics in the Kleisli category $\mathbf{C}_\mathbb{T}$ for a lifting $G_\mathbb{T}$ of G, equivalently given by a distributivity law $\pi : G\mathbb{T} \to \mathbb{T}G$ [9]. A more recent approach, pursued in [13], follows similar lines with TG replaced by GT, the Kleisli category $\mathbf{C}_\mathbb{T}$ by the Eilenberg-Moore category $\mathbf{C}^\mathbb{T}$ and the distributive law $G\mathbb{T} \to \mathbb{T}G$ by a distributive law $\mathbb{T}G \to G\mathbb{T}$. We stick to this latter style of semantics further on.

Definition 7 ($\mathbb{T}$-distributivity). We call G $\mathbb{T}$-*distributive* if there is a distributive law $\pi : \mathbb{T}G \to G\mathbb{T}$, in which case we call π a $\mathbb{T}$-*distributivity* of G.

It is well-known that $\mathbb{T}$-distributivities of G bijectively correspond to liftings $G^\mathbb{T}$ of G to $\mathbf{C}^\mathbb{T}$, which can be expressed by the following commutative diagram

$$\begin{array}{ccc} \mathbf{C}^\mathbb{T} & \xrightarrow{G^\mathbb{T}} & \mathbf{C}^\mathbb{T} \\ {\scriptstyle \mathcal{G}^\mathbb{T}_{\mathrm{EM}}}\downarrow & & \downarrow{\scriptstyle \mathcal{G}^\mathbb{T}_{\mathrm{EM}}} \\ \mathbf{C} & \xrightarrow{G} & \mathbf{C} \end{array} \tag{9}$$

Moreover, as shown in [13], a GT-coalgebra in $\mathbf{C}$ gives rise to a $G^\mathbb{T}$-coalgebra in $\mathbf{C}^\mathbb{T}$ and thus traces of the original coalgebra can be identified as behaviours of the lifted $G^\mathbb{T}$-coalgebra in $\mathbf{C}^\mathbb{T}$. Finally, application of the forgetful functor $\mathcal{G}^\mathbb{T}_{\mathrm{EM}} : \mathbf{C}^\mathbb{T} \to \mathbf{C}$ yields a trace semantics in the original category for, as turns out, $\mathcal{G}^\mathbb{T}_{\mathrm{EM}}$ it sends the final $G^\mathbb{T}$-coalgebra exactly to a final G-coalgebra.

Example 8. [13] A standard example of the presented scenario is given by NA. The language functor L_A is $\mathcal{P}_\omega$-distributive with $\pi : \mathcal{P}_\omega L_A \to L_A\mathcal{P}_\omega$, for every $p \in \mathcal{P}_\omega L_A$ given by

$$\mathsf{pr}_1(\pi(p)) = \exists m.\, \langle \top, m\rangle \in p, \qquad \mathsf{pr}_2(\pi(p)) = \lambda a.\, \{m(a) \mid \langle b, m\rangle \in p\}.$$

It can now be seen by coinduction that the induced transformation of $L_A\mathcal{P}_\omega$-coalgebras to L_A-coalgebras implements automata determinization.

As shown in [13], involving more sophisticated forms of nondeterminism, such as captured by multisets and subdistributions, allows for a treatment of somewhat less standard kinds of machines, such as generative probabilistic systems and weighted automata.

We extend the outlined framework just one step further by introducing the core concept of this paper.

Definition 9 (Real-time $\mathbb{T}$-observer). Given a $\mathbb{T}$-distributive functor G and an endofunctor F, a *real-time $\mathbb{T}$-observer for F*, $F \to G\mathbb{T}$ is given by a natural transformation $\delta : F \to GT$. Disregarding the reference to the original functor F, we call any natural transformations in this format a *generic $\mathbb{T}$-observer*.

The term "real-time" here refers to the fact that the observer depends only on the coalgebra functor, and hence the observation is always performed stepwise with no way to delay until the next step. We elaborate on this further in Section 5.

Essentially, given an observer $F \to G\mathbb{T}$ we can transform any F-coalgebra to a $G\mathbb{T}$-coalgebra and then simply apply the generalised powerset construction to the result. However, we prefer to spell the details as they will be relevant for the remaining presentation.

Note that an observer $\delta : F \to G\mathbb{T}$ gives rise to a natural transformation $\delta_* : TF \to GT$ by the following composition:

$$\delta_* : TF \xrightarrow{T\delta} TGT \xrightarrow{\pi T} GT^2 \xrightarrow{G\mu} GT \tag{10}$$

where π is the $\mathbb{T}$-distributivity of G. The intended use of $\mathbb{T}$-observers is to provide trace semantics for a coalgebra $f : X \to FX$ according to diagram

$$\begin{array}{ccccc} X & \xrightarrow{f} & FX & & \\ {\scriptstyle\eta}\downarrow & & \downarrow{\scriptstyle\eta} & & \\ TX & \xrightarrow{Tf} & TFX & \xrightarrow{\delta_*} & GTX \\ {\scriptstyle\hat{s}}\downarrow & \overset{s}{\cdots\cdots\nearrow} & & & \downarrow{\scriptstyle G\hat{s}} \\ \nu G & & \xrightarrow{\iota} & & G\nu G \end{array} \tag{11}$$

Here, ι is the final coalgebra structure, $s = \delta_*(Tf)$ and $\hat{s}$ is the universal arrow induced by s. Given some $a : 1 \to X$ we call $\hat{s}\eta a$ the *δ-trace* of f at a. This naturally generalizes the construction from [13] by taking $F := GT$ and $\delta := (G\mu)\pi$. The DA and NA examples can now be treated on the same footing.

Example 10. Let $FX = 2 \times (\mathcal{P}_\omega X)^A$ be the NA-functor as in Example 8. The generalised powerset construction from [13] amounts to the identity $\mathcal{P}_\omega$-observer $F \simeq L_A \mathcal{P}_\omega$ and therefore yields the expected trace semantics w.r.t. L_A.

In case of the DA-functor $FX = 2 \times X^A$ we take as the $\mathcal{P}_\omega$-observer $L_A(\eta_X) : 2 \times X^A \to L_A \mathcal{P}_\omega X$ where η_X is the unit of $\mathcal{P}_\omega$ and obtain the conventional trace semantics, which does, of course, coincide with the canonical behavioural equivalence.

Proposition 11. *To give an observer $\delta : F \to G\mathbb{T}$ is the same as to give a natural transformation for the following pasting diagram:*

$$\begin{array}{ccc} \mathbf{C} & \xrightarrow{F} & \mathbf{C} \\ {\scriptstyle \mathcal{F}^{\mathbb{T}}_{\mathrm{EM}}}\downarrow & \swarrow & \downarrow{\scriptstyle \mathcal{F}^{\mathbb{T}}_{\mathrm{EM}}} \\ \mathbf{C}^{\mathbb{T}} & \xrightarrow{G^{\mathbb{T}}} & \mathbf{C}^{\mathbb{T}} \end{array} \tag{12}$$

5 Lookahead Observers

We introduced the notion of a $\mathbb{T}$-observer as a fairly modest generalization of an existing device. Here, we would like to argue that this notion is in fact derivable from some rather general category-theoretic considerations.

Let us consider a pair of categories $\mathbf{C}$, $\mathbf{D}$, a pair of endofunctors F, G over them and the corresponding categories of coalgebras $\mathsf{coalg}_F(\mathbf{C})$, $\mathsf{coalg}_H(\mathbf{D})$. The idea is: objects of $\mathsf{coalg}_F(\mathbf{C})$ represent original systems of interest, while objects of $\mathsf{coalg}_H(\mathbf{D})$ represent systems of observable behaviours of the latter. It appears reasonable to capture such kind of an observation scenario by a pair of functors $V : \mathbf{C} \to \mathbf{D}$ and $\widehat{V} : \mathsf{coalg}_F(\mathbf{C}) \to \mathsf{coalg}_H(\mathbf{D})$ such that $\widehat{V}$ is a lifting of V, in other words the diagram

$$\begin{array}{ccc} \mathsf{coalg}_F(\mathbf{C}) & \xrightarrow{\widehat{V}} & \mathsf{coalg}_H(\mathbf{D}) \\ {\scriptstyle U_F}\downarrow & & \downarrow{\scriptstyle U_G} \\ \mathbf{C} & \xrightarrow{V} & \mathbf{D} \end{array} \tag{13}$$

with U_F, U_G being the evident forgetful functors, commutes.

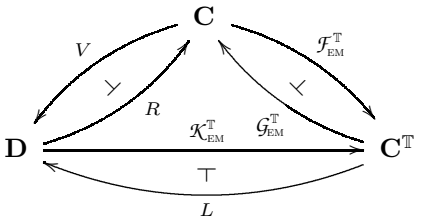
Suppose U_F has a right adjoint R_F. This gives rise to a *cofree comonad F^∞ on* F, explicitly, $F^\infty X = \nu\gamma.(F\gamma \times X)$ (see e.g. [30]). Since $\mathsf{coalg}_F(\mathbf{C})$ is isomorphic to the Eilenberg-Moore category $\mathsf{Coalg}_{F^\infty}(\mathbf{C})$, an endofunctor $\widehat{V}$ on $\mathsf{coalg}_F(\mathbf{C})$ is equivalently an endofunctor on $\mathsf{Coalg}_{F^\infty}(\mathbf{C})$. Furthermore, we have

Lemma 12. *Provided F^∞ exists, to give a lifting $\widehat{V}$ for V is the same as to give a natural transformation*

$$VF^\infty \to HV. \tag{14}$$

Let us assume that the functor $V : \mathbf{C} \to \mathbf{D}$ in (13) has a right adjoint $R : \mathbf{D} \to \mathbf{C}$ and let us therefore obtain a monad $T = RV$ on $\mathbf{C}$. Let $\mathcal{K}^{\mathbb{T}}_{\mathrm{EM}} : \mathbf{D} \to \mathbf{C}^{\mathbb{T}}$ be the

induced comparison functor. Suppose moreover that $\mathcal{K}^{\mathbb{T}}_{\mathrm{EM}}$ has a left adjoint L so that, in summary, we have the following picture:



where $\mathcal{F}^{\mathbb{T}}_{\mathrm{EM}} = \mathcal{K}^{\mathbb{T}}_{\mathrm{EM}} V$ and $R = \mathcal{G}^{\mathbb{T}}_{\mathrm{EM}} \mathcal{K}^{\mathbb{T}}_{\mathrm{EM}}$. Since a composition of left (right) adjoints is again a left (right) adjoint and a left (right) adjoint to a functor is uniquely defined, also $V = L\mathcal{F}^{\mathbb{T}}_{\mathrm{EM}}$ and $\mathcal{G}^{\mathbb{T}}_{\mathrm{EM}} = RL$.

Theorem 13. *In the situation* (13)*, suppose (i) U_F has a right adjoint R_F, (ii) V has a right adjoint R and (iii) the comparison functor $\mathcal{K}^{\mathbb{T}}_{\mathrm{EM}}$ a has a left adjoint L. Let H be an endofunctor $\mathbf{D} \to \mathbf{D}$ such that $GR \simeq RH$ with some $G : \mathbf{C} \to \mathbf{C}$. Then*

1. *$\mathcal{K}^{\mathbb{T}}_{\mathrm{EM}} HL : \mathbf{C}^{\mathbb{T}} \to \mathbf{C}^{\mathbb{T}}$ is a lifting of G to $\mathbf{C}^{\mathbb{T}}$ and thus G is $\mathbb{T}$-distributive;*
2. *the rule sending any $\alpha : F^\infty \to GT$ to*

$$VF^\infty \xrightarrow{V\alpha} VGRV \xrightarrow{V\rho V} VRHV \xrightarrow{\epsilon HV} HV$$

where ρ is the isomorphism $HV \simeq VG$ and ϵ is the counit of the adjunction, determines a one-to-one correspondence between $\mathbb{T}$-observers $F^\infty \to G\mathbb{T}$ and natural transformations (14)*.*

Remark 14. While the conditions (i) and (iii) of Theorem 13 seem to be relatively mild (e.g. (i) is fulfilled when F is accessible, (iii) is fulfilled when $\mathbf{D}$ has coequalizers [1]), condition (ii) is essential. A somewhat unexpected source of situations (13) satisfying (ii) are *logical connections* [20] used in coalgebraic modal logic. These can be related to our view by instantiating $\mathbf{D}$ with $\mathbf{E}^{op}$ for some $\mathbf{E}$; Coalgebras over $\mathbf{D}$ would then be algebras over $\mathbf{E}$. A detailed analysis of the relation between observers and logical connections is subject to further work.

Theorem 13 inspires the following definition.

Definition 15 (Lookahead $\mathbb{T}$-observer). A *lookahead $\mathbb{T}$-observer for F* is a real-time $\mathbb{T}$-observer for F^∞.

For every F-coalgebra $f : X \to FX$ we can form a F^∞-coalgebra $f' : X \to F^\infty X$ as the final arrow from $\langle f, \mathrm{id}\rangle : X \to FX \times X$ to $F^\infty X$. In order to obtain a δ-trace for f by means of a lookahead observer $\delta : F^\infty \to GT$ we just apply the construction (11) to f' instead of f.

We clarify the distinction between real-time and lookahead observers by a small example inspired by [14].

Example 16 (Infinite streams). Consider the following, perhaps slightly artificial, problem: Given an infinite stream over a set L, calculate the stream obtained by missing out all those maximal finite segments of it, which consist of a specified element $a \in L$, e.g. if the whole stream is an infinite repetition of a then the original stream is returned. The functor $FX = L \times X$ is known to generate infinite streams exactly as we need. A real-time observer would then have the format $L \times \mathrm{Id} \to L \times T$, which is not suitable for it would not give a way to access the next element of the stream as required unless the front one is a.

By comparison, consider a lookahead observer $skip_a : F^\infty \to F$ w.r.t. the identity monad. Note that $F^\infty X \simeq \nu\gamma.\,(F\gamma \times X) \simeq (L \times X)^\infty$, i.e. $F^\infty X$ consists of infinite streams over $L \times X$. Hence we define

$$skip_a[\langle a, x_1\rangle, \langle a, x_2\rangle, \ldots, \langle a, x_n\rangle, \langle b, x_{n+1}\rangle, \ldots] = \langle b, x_{n+1}\rangle,$$
$$skip_a[\langle a, x_1\rangle, \langle a, x_2\rangle, \ldots, \langle a, x_n\rangle, \langle a, x_{n+1}\rangle, \ldots] = \langle a, x_2\rangle.$$

A coalgebra $f : X \to FX$ gives rise to a coalgebra $f' : X \to F^\infty X$ so that a behaviour $[a_1, a_2, \ldots]$ of the original system carried out by a state x_1 maps to a behaviour $[\langle a_1, x_1\rangle, \langle a_2, x_2\rangle, \ldots]$ of the transformed system where the second components of the tuples protocol the intermediate states visited. It is now easy to see that a $skip_a$-trace of f at $x : 1 \to X$ is obtained from the corresponding stream of behaviours as expected.

As noted in [14], examples like $skip_a$, considered as such, carry a little of operational meaning, which fact indicates that restricting to real-time observers can well be a reasonable idea. However, as we shall see, allowing for a lookahead pays off as it immediately enables useful coalgebraic meta-constructions for a small added price.

6 Internal Actions

Consider the language functor $L_{A+E}X = 2 \times X^{A+E}$ where the actions are partitioned into visible A and internal ones E. Applying the standard finality argument produces traces over the whole set $A+E$, which can be undesirable because of the presence of internal actions. A real-time observer can not be helpful to tackle this issue precisely because if we run into an internal action we must drop it and hold up the output until a non-internal one occurs. As one would expect, a suitable lookahead observer carries the necessary effect.

Let us note first that $L^\infty_{A+E}X \simeq \nu\gamma.\,(2 \times X \times \gamma^{A+E}) \simeq (2 \times X)^{(A+E)^*}$. Then we define a lookahead observer $\delta_X : L^\infty_{A+E}X \to \mathcal{P}_{\omega 1}L_A X$ by the expression

$$\delta\langle o, t\rangle = \{\langle o(w), \lambda a.\, t(w \cdot a)\rangle \mid w \in E^*\}$$

Intuitively, we form partial runs of the original automaton over $A + E$ corresponding to the trace prefixes of the form $\tau_1, \tau_2, \ldots, \tau_n, a$ with $\tau_i \in E$, $a \in A$ and for every such run construct a one-step a-transition of the target automaton. As it must, the target automaton becomes highly nondeterministic, hence the

use of the countable powerset functor $\mathcal{P}_{\omega 1}$. Determinization is ensured on the spot by defining a $\mathcal{P}_{\omega 1}$-distributivity for L_A as in Example 8.

It is not difficult to instantiate the presented idea to capture weak bisimilarity of processes by reduction to strong bisimilarity [17]. Unfortunately, unlike the case of linear traces, considered above, it appears to be impossible to eliminate the internal action. This seems to be a consequence of the standard fact that weak bisimilarity fails to be a congruence. However, we can make use of the fact that two processes are weakly bisimilar iff they become strongly bisimilar after saturating their state transitions by adjoining pre- and postfixes of chained internal actions [6, 17]. Informally, this amounts to creating one a-transition in the target system corresponding to a chain of transitions of the original system labelled by τ^n, a, τ^m if $a \neq \tau$, and one internal transition corresponding to a chain of transitions labelled by τ^n. Here, τ^n and τ^m denote finite, possibly empty, chains of the internal action τ.

Let $FX = \mathcal{P}_\omega(X)^{A_\tau}$ be the functor of finitely-branching labelled transition systems over finite $A_\tau = A \cup \{\tau\}$ [26]. Note that $F^\infty X \simeq \nu\gamma.\,(\mathcal{P}_\omega(\gamma)^{A_\tau} \times X)$. We define a lookahead observer $\pi_X : \nu\gamma.\,(\mathcal{P}_\omega(\gamma)^{A_\tau} \times X) \to \mathcal{P}_{\omega 1}(X)^{A_\tau}$ for F w.r.t. the identity monad by expression:

$$\pi(t_0)(a) = \bigcup\{\mathsf{pr}_2(t_n)(a_n) \mid t_1 \in \mathsf{pr}_1(t_0)(a_0), \ldots, t_n \in \mathsf{pr}_1(t_{n-1})(a_{n-1})\}$$

where the union is taken over all sequences $a_1, \ldots, a_n$ of the form $\tau, \ldots, \tau, a, \tau, \ldots, \tau$ if $a \neq \tau$ and $\tau, \ldots, \tau$ is $a = \tau$.

The presented construction is the most straightforward one. As a result, the obtained transition system receives an enormous number of junk τ-transitions. A further optimisation for practical purposes should not be difficult, but would demand for involving more sophisticated versions of the observer.

7 Push-Down Automata

We treat push-down automata analogously to NA by using the functor $FX = \mathcal{P}_\omega(S^*) \times \mathcal{P}_\omega(X \times S^*)^{A \times S}$ where A stands for the input alphabet and S stands for the set of stack symbols. Given a finite set X, a coalgebra $\langle o, t\rangle : X \to FX$ captures the automaton carrying the following data:

Finite Set of Accepting Configurations $Acc \subseteq X \times S^*$ consisting of all such pairs $\langle x, w\rangle \in X \times S^*$ that $o(x)(w) = 1$; Common possible choices for Acc include [25]: $\{\langle x, w\rangle \mid x \in Fin\}$, $\{\langle x, w\rangle \mid w = \epsilon\}$, $\{\langle x, w\rangle \mid x \in Fin, w = \epsilon\}$, and $\{\langle x, w\rangle \mid w = sw', s \in S'\}$ where $Fin \subseteq X$ is a distinguished set of final states, $S' \subseteq S$ is a distinguished set of stack symbols, and ϵ denotes the empty stack.

Transition Function $X \times A \times S \to \mathcal{P}_\omega(X \times S^*)$ obtained by uncurrying t; Here, the elements of an input triple $\langle x, a, s\rangle$ refer to a current state x, an input alphabet symbol a and the current top stack symbol s, subject to removal. The outcome of the transition function is a finite set of pairs $\langle x', w\rangle$ where x' is the new state and w is a string of elements to be pushed onto the stack.

As already noticed in [28], transitions of a push-down automaton can be considered as a nondeterministic side-effecting function w.r.t. the set of stacks S^*

as underlying store, which suggests using the monad $\mathcal{P}_\omega(X \times S^*)^{S^*}$. Here, we propose a nondeterministic version of the stack monad from Section 3 as a better choice, for—as we have shown in Proposition 5—it is directly generated by the stack operations. Let $\mathbb{T}$ be that monad henceforth.

Note that T does not appear as a subexpression of the expression defining F. More importantly, operations encoded by F are only commands to be applied to a stack and not ready stack transformers.

Due to an isomorphism $FX \simeq \mathcal{P}_\omega(S^*) \times \mathcal{P}_\omega(X \times (S \times A \times S^*))$, behaviours of F can be understood as finitely-branching trees, modulo bisimilarity, whose nodes are annotated by finite sets of stacks and whose branches are labelled with commands of either of two forms: $(a; s/s_1 \dots s_n)$, $(s/s_1 \dots s_n)$.

Let $\alpha_X : \mathcal{P}_\omega(X \times S^*)^S \to TX$ be the natural transformation defined by the clauses

$$\alpha(f)(\epsilon) = \emptyset, \qquad \alpha(f)(s \cdot w) = \{\langle x, u \cdot w\rangle \mid \langle x, u\rangle \in f(s)\}.$$

This gives rise, in an obvious way, to a real-time $\mathbb{T}$-observer $\delta_X : FX \to \mathcal{P}(S^*) \times (TX)^A$ for F. Finally, we endow $\mathcal{P}(S^*) \times -^A$ with a $\mathbb{T}$-distributivity $\pi_X : T(\mathcal{P}(S^*) \times X^A) \to \mathcal{P}(S^*) \times (TX)^A$ by the equations:

$$\mathsf{pr}_1(\pi(p)) = \{w \in S^* \mid \langle w'', m, w'\rangle \in p(w), w'' \cap w' \neq \emptyset\},$$
$$\mathsf{pr}_2(\pi(p)) = \lambda a.\, \lambda w.\, \{m(a) \mid \langle b, m\rangle \in p(w)\},$$

which is in a perfect correspondence with [28]. The resulting traces are the elements of the final coalgebra $\nu\gamma.\, (\mathcal{P}(S^*) \times X^A) \simeq \mathcal{P}(A^*)^{S^*}$ and are indeed maps from the set of initial stack values to languages over A.

According to the standard definition (e.g. [25]), A has form $A' + 1$ where the adjoined element can be viewed as a special symbol for internal transitions (otherwise the automaton is called *real-time* and captures precisely non-empty context-free languages over A [25]), which results in the undesirable effect of having the internal symbol in the traces. A solution to that would be to involve a lookahead observer as in Section 6, which we do not spell out here.

8 Reactive Programs with Generic Side-Effects

Reactive coalgebra-based systems over generic side-effects encapsulated by a monad appear in the literature in the form of coalgebraic component-based frameworks [3] or as generic calculi for side-effecting processes [7]. Here, we sketch the perspectives of defining observation-based semantics for systems of this kind.

Let (for simplicity) $\mathbf{C} = \mathbf{Set}$ and consider coalgebras of the form

$$f : X \to T(O \times X)^I \tag{15}$$

where T is a functorial part of a monad $\mathbb{T}$, capturing some generic side-effect, I is an input type, O is an output type. One way to look at a system of this kind is as a generalized Mealy machine [28]—the latter would be obtained by instantiating $\mathbb{T}$ with the identity monad.

Note that $\nu\gamma.\,(O \times \gamma)^I \simeq \nu\gamma.\,(O^I \times \gamma^I) \simeq (O^I)^{I^*} = O^{I^+}$ where I^+ stands for nonempty lists over I. It is easy to see by induction that the function space $I^+ \to O$ can be considered as a subspace of stream transformers $I^\omega \to O^\omega$ formed by so-called *causal maps* [27]: a map $t : I^\omega \to O^\omega$ is *causal* if $t(s)(n) = t(s')(n)$ whenever $s(i) = s'(i)$ for all $i \leq n$; In other words, the n-th element of the output stream $t(s)$ depends only on the elements of s in the positions from the first one up to the n-th. The latter definition of causality is the standard one, however, the former one is more suitable for our generalization, which is as follows.

Definition 17 ($\mathbb{T}$-causality). Given a strong monad $\mathbb{T}$ over a category $\mathbf{C}$ with finite products and coproducts such that any initial algebra $X^* = \mu\gamma.\,(1 + X \times \gamma)$ exists, we call a morphism $t : I^* \to TO^*$ $\mathbb{T}$-*causal* if the diagram

$$\begin{array}{ccc} I^* & \xrightarrow{\quad t \quad} & TO^* \\ {\scriptstyle \iota_I}\downarrow & & \downarrow{\scriptstyle T\iota_O} \\ 1 + I \times I^* & & T(1 + O \times O^*) \\ {\scriptstyle \mathrm{id} + \mathsf{pr}_2}\downarrow & & \downarrow{\scriptstyle T(\mathrm{id} + \mathsf{pr}_2)} \\ 1 + I^* & \xrightarrow{[\eta\,\mathsf{inl}, (T\,\mathsf{inr})t]} & T(1 + O^*) \end{array}$$

commutes where $\iota_X : X^* \to 1 + X \times X^*$ is the initial algebra structure on X^*.

It can be readily verified by induction that for $\mathbf{C} = \mathbf{Set}$ with $\mathbb{T}$ being the identity monad $\mathbb{T}$-causality agrees with the standard definition.

We now proceed with defining a trace semantics for coalgebras of type (15). Let $\mathbb{R}$ be the monad with the functorial part $RX = T(O^* \times X)$ and monadic operations induced by the obvious monoidal structure on O^*. Then we define a real-time $\mathbb{R}$-observer $\delta_X : T(O \times X)^I \to TO^* \times RX^I$ and an $\mathbb{R}$-distributivity $\pi_X : R(TO^* \times X^I) \to TO^* \times RX^I$ by the expressions

$$\delta(t) = \langle \eta\alpha, \lambda x.\, T(\beta \times \mathrm{id})(t(x)) \rangle, \tag{16}$$

$$\pi(t) = \Big(\mathsf{let}\, r = T(\gamma \times \mathrm{id})\mu(T\tau)t \;\; \mathsf{in}\, \langle (T\,\mathsf{pr}_1)r, \lambda x.\, T(\mathrm{id} \times at(x))r \rangle\Big) \tag{17}$$

where $\alpha : 1 \to O^*$, $\beta : O \to O^*$, $\gamma : O^* \times O^* \to O^*$ are the obvious operations for creating the empty list, forming a single-element list from a given element and for list concatenation correspondingly; τ is the strength of $\mathbb{T}$; $at(x) = \lambda f.\, f(x)$; the intermediate value r has $T(O^* \times X^I)$ as the output type. This induces traces for (15) as elements of $\nu\gamma.\,(TO^* \times \gamma^I) \simeq (TO^*)^{I^*}$.

The idea behind the monad $\mathbb{R}$ is to collect the outputs from O while composing the computational effects between iterations. The strength τ of the monad $\mathbb{T}$ plays a crucial role in this process, as it allows for propagating the output values downwards through the layers of $\mathbb{T}$-computations. We can now relate to $\mathbb{T}$-causality as follows.

Proposition 18. *Traces induced by* (16)–(17) *are* $\mathbb{T}$*-causal.*

9 Conclusions

We have laid down the foundations of a generic monad-based notion of observation for systems represented by coalgebras. This notion establishes a bridge

between an implementation of a system and its observable behaviour, and therefore provides a basic abstraction idiom for studying systems from the most general perspective. Whereas, from the technical point of view, our development is only a mild modification of the existing machinery, as we have shown, our notion of observation has a universal character—under reasonable assumptions, it arises from a fairly basic observation scenario. Every observer in our framework comes with an associated generalized notion of trace, which can vary over a large spectrum of equivalences refining the canonical notion of bisimilarity of the original system. We have shown how the introduced notion of observation allows for a smooth and concise treatment of such traditionally delicate examples as internal actions, weak bisimilarity and pushdown automata. As a further improvement of the latter case we have presented two versions of a stack monad and operational characterizations for them in the spirit of Plotkin and Power.

Related Work. This work descends from [13] as well as from the more recent [28] where the generalized powerset construction was introduced. Natural transformations of the form $TF \to GT$, which one can treat as a form of a $\mathbb{T}$-observer occasionally appear in [13] for sakes of relating Kleisli and Eilenberg-Moore styles of semantics. Insufficient expressivity of the standard distributive law argument has been acknowledged in [29] in the specific case of the finitary subdistribution monad D_ω combined with the functor $1 + A \times -$: $D_\omega(1 + A \times X)$ only embeds into $[0, 1] \times (D_\omega X)^A$ (and thus can be naturally regarded as an observer), unlike the analogous case with $\mathcal{P}_\omega$ instead of D_ω when an isomorphism would take place.

Future Work. Format constraints applied to this note do not give a chance to develop the presented theory to a sufficient extent. We would like to develop this further along the following lines.

– *Formal languages and machines.* One objective of this paper was to improve the coalgebraic treatment of push-down automata. We believe that an analogous treatment of Turing machines should not be difficult, except that instead of classical Turing machines one should involve *reactive Turing machines* presented in [2] as a more coalgebra-friendly concept.

– *Coalgebraic modal logic.* As indicated in Remark 14, there is a technical relation between observation scenarios (13) and logical connections. In view of the classical relation between testing and observational semantics this may be not a coincidence and should be studied in detail; The notion of a lookahead observer may also suggest a way for a logical characterisation of the weak bisimulation, by involving F^∞-coalgebras derived from F-coalgebras.

– *Rational fixpoints of functors.* The notion of a regular language is known to be nicely captured coalgebraically by a *rational fixpoint* ρL_A of the language functor L_A by Milius and collaborators (see e.g. [4]). As expected, ρL_A sits inside the greatest fixpoint νL_A of all formal languages. The question is, if the theory of rational fixpoints can be extended (somehow) to other kinds of machines under the presented observational treatment, e.g. to push-down automata.

– *Notion of observation.* We intend to study compositions of observation scenarios (13), such as sequential and parallel ones, and their impact on the monads underlying the corresponding observers. As we have seen, the notion of trace

equivalence, induced by an observer can be as fine as the usual bisimilarity. We anticipate the introduction of a class of observers capturing linearity of traces and a construction for generating such observers from a functor and a monad in a universal manner analogously to forming tensors of monads with powersets [8]. Theorem 13 indicates that it might be reasonable to relax the definition of an observer by rebasing it on pairs of adjoint functors and dropping $\mathbb{T}$-distributivity.

Acknowledgements. The author wishes to thank Stefan Milius for a brief but definite update on distributive laws for comonads. Thanks to Alexandra Silva, Daniel Hausmann and anonymous referees for their valuable feedback.

References

[1] Adámek, J., Herrlich, H., Strecker, G.: Abstract and concrete categories. John Wiley & Sons Inc., New York (1990)
[2] Baeten, J.C.M., Luttik, B., van Tilburg, P.: Reactive turing machines. In: Owe, O., Steffen, M., Telle, J.A. (eds.) FCT 2011. LNCS, vol. 6914, pp. 348–359. Springer, Heidelberg (2011)
[3] Barbosa, L.S.: Towards a calculus of state-based software components. Journal of Universal Comp. Sci. 9, 891–909 (2003)
[4] Bonsangue, M.M., Milius, S., Silva, A.: Sound and complete axiomatizations of coalgebraic language equivalence. ACM Trans. Comp. Logic 14(1), 7:1–7:7 (2013)
[5] Dubuc, E.: Kan Extensions in Enriched Category Theory. LNM, vol. 145 (1970)
[6] Fiore, M., Cattani, G.L., Winskel, G.: Weak bisimulation and open maps. In: LICS 1999 (1999)
[7] Goncharov, S., Schröder, L.: A coinductive calculus for asynchronous side-effecting processes. In: Owe, O., Steffen, M., Telle, J.A. (eds.) FCT 2011. LNCS, vol. 6914, pp. 276–287. Springer, Heidelberg (2011)
[8] Goncharov, S., Schröder, L.: Powermonads and tensors of unranked effects. In: LICS 2011, pp. 227–236 (2011)
[9] Hasuo, I., Jacobs, B., Sokolova, A.: Generic trace theory. In: CMCS 2006. Elect. Notes in Theor. Comp. Sci., vol. 164, pp. 47–65. Elsevier (2006)
[10] Hyland, M., Plotkin, G., Power, J.: Combining computational effects: Commutativity & Sum. In: TCS 2002, vol. 223, pp. 474–484. Kluwer (2002)
[11] Jacobs, B.: Trace semantics for coalgebras. Electron. Notes Theor. Comput. Sci. 106, 167–184 (2004)
[12] Jacobs, B., Poll, E.: Coalgebras and Monads in the Semantics of Java. Theoret. Comput. Sci. 291, 329–349 (2003)
[13] Jacobs, B., Silva, A., Sokolova, A.: Trace semantics via determinization. In: Pattinson, D., Schröder, L. (eds.) CMCS 2012. LNCS, vol. 7399, pp. 109–129. Springer, Heidelberg (2012)
[14] Klin, B.: Bialgebras for structural operational semantics: An introduction. Theor. Comput. Sci. 412(38), 5043–5069 (2011)
[15] Kock, A.: Strong functors and monoidal monads. Archiv der Mathematik 23(1), 113–120 (1972)
[16] Mac Lane, S.: Categories for the Working Mathematician. Springer (1971)
[17] Milner, R.: Communication and concurrency. Prentice-Hall, Inc., Upper Saddle River (1989)
[18] Moggi, E.: A modular approach to denotational semantics. In: Curien, P.-L., Pitt, D.H., Pitts, A.M., Poigné, A., Rydeheard, D.E., Abramsky, S. (eds.) CTCS 1991. LNCS, vol. 530, pp. 138–139. Springer, Heidelberg (1991)

[19] Moggi, E.: Notions of computation and monads. Inf. Comput. 93, 55–92 (1991)
[20] Pavlovic, D., Mislove, M., Worrell, J.B.: Testing semantics: Connecting processes and process logics. In: Johnson, M., Vene, V. (eds.) AMAST 2006. LNCS, vol. 4019, pp. 308–322. Springer, Heidelberg (2006)
[21] Plotkin, G., Power, J.: Adequacy for algebraic effects. In: Honsell, F., Miculan, M. (eds.) FOSSACS 2001. LNCS, vol. 2030, pp. 1–24. Springer, Heidelberg (2001)
[22] Plotkin, G., Power, J.: Notions of computation determine monads. In: Nielsen, M., Engberg, U. (eds.) FOSSACS 2002. LNCS, vol. 2303, pp. 342–356. Springer, Heidelberg (2002)
[23] Plotkin, G., Power, J.: Algebraic operations and generic effects. Appl. Cat. Struct. 11, 69–94 (2003)
[24] Power, J., Shkaravska, O.: From comodels to coalgebras: State and arrays. In: CMCS 2004. ENTCS, vol. 106, pp. 297–314 (2004)
[25] Rozenberg, G., Salomaa, A. (eds.): Handbook of formal languages. Word, Language, Grammar, vol. 1. Springer-Verlag New York, Inc. (1997)
[26] Rutten, J.: Universal coalgebra: A theory of systems. Theoret. Comput. Sci. 249, 3–80 (2000)
[27] Rutten, J.J.M.M.: Algebraic specification and coalgebraic synthesis of mealy automata. Electr. Notes Theor. Comput. Sci. 160, 305–319 (2006)
[28] Silva, A., Bonchi, F., Bonsangue, M., Rutten, J.: Generalizing determinization from automata to coalgebras. LMCS 9(1) (2013)
[29] Silva, A., Sokolova, A.: Sound and complete axiomatization of trace semantics for probabilistic systems. Electr. Notes Theor. Comput. Sci. 276, 291–311 (2011)
[30] Uustalu, T., Vene, V.: Comonadic notions of computation. Electron. Notes Theor. Comput. Sci. 203(5), 263–284 (2008)

Full Abstraction for Fair Testing in CCS

Tom Hirschowitz*

CNRS and Université de Savoie

Abstract. In previous work with Pous, we defined a semantics for CCS which may both be viewed as an innocent presheaf semantics and as a concurrent game semantics. It is here proved that a behavioural equivalence induced by this semantics on CCS processes is fully abstract for fair testing equivalence.

The proof relies on a new algebraic notion called *playground*, which represents the 'rule of the game'. From any playground, two languages, equipped with labelled transition systems, are derived, as well as a strong, functional bisimulation between them.

Keywords: Programming languages, categorical semantics, presheaf semantics, game semantics, concurrency, process algebra.

1 Introduction

Motivation and Previous Work. Innocent game semantics, invented by Hyland and Ong [20], led to fully abstract models for a variety of functional languages, where programs are interpreted as strategies in a game. Presheaf models [22, 6] were introduced by Joyal et al. as a semantics for process algebras, in particular Milner's CCS [28]. Previous work with Pous [19] (HP) proposes a semantics for CCS, which reconciles these apparently very different approaches. Briefly, (1) on the one hand, we generalise innocent game semantics to both take seriously the possibility of games with more than two players and consider strategies which may accept plays in more than one way; (2) on the other hand, we refine presheaf models to take parallel composition more seriously. This leads to a model of CCS which may both be seen as a concurrent game semantics, and as an innocent presheaf model, as we now briefly recall.

To see that presheaf models are a concurrent, non-innocent variant of game semantics, recall that the base category, say $\mathbb{C}$, for such a presheaf model typically has as objects sequences of labels, or configurations in event structures, morphisms being given by prefix inclusion. Such objects may be understood as plays in some game. Now, in standard game semantics, a strategy is a prefix-closed (non-empty) set of plays. Unfolding the definition, this is the same as a functor $\mathbb{C}^{op} \to 2$, where 2 is the poset category $0 \leqslant 1$: the functor maps a play to 1 when it is accepted by the strategy, and to 0 otherwise. It is known since

* Partially funded by the French ANR projet blanc "Formal Verification of Distributed Components" PiCoq ANR 2010 BLAN 0305 01 and CNRS PEPS CoGIP.

R. Heckel and S. Milius (Eds.): CALCO 2013, LNCS 8089, pp. 175–190, 2013.

Harmer and McCusker [15] that this notion of strategy does not easily adapt to non-determinism or concurrency. Presheaf semantics only slightly generalises it by allowing strategies to accept a play in several ways. A strategy S now maps each play p to a *set* $S(p)$. The play is accepted when $S(p)$ is non-empty, and, because there are then no functions $S(p) \to \varnothing$, being accepted remains a prefix-closed property of plays. The passage from 2 to more general sets allows to express branching-time semantics.

This links presheaf models with game models, but would be of little interest without the issue of *innocence*. Game models, indeed, do not always accept *any* prefix-closed set of plays S as a strategy: they demand that any choice of move in S depends only on its *view*. E.g., consider the CCS process $P = (a|(b \oplus c))$, where $\oplus$ denotes internal choice, and a candidate strategy accepting the plays $\epsilon, (a), (b), (c), (ab)$, but not (ac). This strategy refuses to choose c after a has been played. Informally, there are two players here, one playing a and the other playing $b \oplus c$; the latter should have no means to know whether a has been played or not. We want to rule out this strategy on the grounds that it is not innocent.

Our technical solution for doing so is to refine the notion of play, making the number of involved players more explicit. Plays still form a category, but they admit a subcategory of *views*, which represent a single player's possible perceptions of the game. This leads us to two equivalent categories of strategies. In the first, strategies are presheaves on views. In the second category, strategies are certain presheaves on arbitrary plays, satisfying an innocence condition. Parallel composition, in the game semantical sense, is best understood in the former category: it merely amounts to copairing. Parallel composition, in the CCS sense, which in standard presheaf models is a complex operation based on some labelling of transitions or events, is here just a move in the game. The full category of plays is necessary for understanding the global behaviour of strategies. It is in particular needed to define our semantic variant of fair testing equivalence, described below. One may think of presheaves on views as a syntax, and of innocent presheaves on plays as a semantics. The combinatorics of passing from local (views) to global (arbitrary plays) are dealt with by right Kan extension.

Discussion of Main Results. In this paper, we further study the semantics of HP, to demonstrate how close it is to operational semantics. For this, we provide two results. The most important, in the author's view, is full abstraction w.r.t. *fair testing semantics*. But the second result might be considered more convincing by many: it establishes that our semantics is fully abstract w.r.t. weak bisimilarity. The reason why it is here considered less important is that it relies on something external to the model itself, namely an LTS for strategies, constructed in an *ad hoc* way. Considering that a process calculus is defined by its reduction semantics, rather than by its possibly numerous LTSs, testing equivalences, which rely on the former, are more intrinsic than various forms of bisimilarity.

Now, why consider fair testing among the many testing equivalences? First of all, let us mention that we could probably generalise our result to any reasonable

testing equivalence. Any testing equivalence relies on a 'testing predicate' $\perp$. E.g., for fair testing, it is the set of processes from which any unsuccessful, finite reduction sequence extends to a successful one. We conjecture that for any other predicate $\perp'$, if $\perp'$ is stable under weak bisimilarity, i.e, $P \simeq Q \in \perp'$ implies $P \in \perp'$, then we may interpret the resulting equivalence in terms of strategies, and get a fully abstract semantics. However, this paper is already quite complicated, and pushes generalisation rather far in other respects (see below). We thus chose to remain concrete about the considered equivalence. It was then natural to consider fair testing, as it is both one of the most prominent testing equivalences, and one of the finest. It was introduced independently by Natarajan and Cleaveland [30], and by Brinksma et al. [3, 33] (under the name of *should* testing in the latter paper), with the aim of reconciling the good properties of observation congruence [29] w.r.t. divergence, and the good properties of previous testing equivalences [7] w.r.t. choice. Typically, $a.b + a.c$ and $a.(b \oplus c)$ (where $+$ denotes guarded choice and $\oplus$ denotes internal choice) are not observation congruent, which is perceived as excessive discriminating power of observation congruence. Conversely, $(!\tau) \mid a$ and a are not must testing equivalent, which is perceived as excessive discriminating power of must testing equivalence. Fair testing rectifies both defects, and has been the subject of further investigation, as summarised, e.g., in Cacciagrano et al. [5].

Overview. We now give a bit more detail on the contents, warning the reader that this paper is only an extended abstract, and that more technical details may be found in a (submitted) long version [18]. After recalling the game from HP in Section 2, as well as strategies and our semantic fair testing equivalence $\sim_f$ in Section 3, we prove that the translation $(\!|-|\!)$ of HP from CCS to strategies is such that $P \sim_{f,s} Q$ iff $(\!|P|\!) \sim_f (\!|Q|\!)$, where $\sim_{f,s}$ is standard fair testing equivalence (Theorem 4.6).

Our first attempts at proving this where obscured by easy, yet lengthy case analyses over moves. This prompted the search for a way of factoring out what holds 'for all moves'. The result is the notion of *playground*, surveyed in Section 4.1. It is probably not yet in a mature state, and hopefully the axioms will simplify in the future. We show how the game recalled above organises into such a playground $\mathbb{D}^{CCS}$. We then develop the theory in Section 4.2, defining, for any playground $\mathbb{D}$, two LTSs, $\mathcal{T}_{\mathbb{D}}$ and $\mathcal{S}_{\mathbb{D}}$, of *process terms* and *strategies*, respectively, over an alphabet $\mathbb{F}_{\mathbb{D}}$. We then define a map $[\![-]\!] \colon \mathcal{T}_{\mathbb{D}} \to \mathcal{S}_{\mathbb{D}}$ between them, which we prove is a strong bisimulation.

Returning to the case of CCS in Section 4.3, we obtain that $\mathcal{S}_{\mathbb{D}^{CCS}}$ indeed has strategies as states, and that $\sim_f$ may be characterised in terms of this LTS. Furthermore, unfolding the definition of $\mathcal{T}_{\mathbb{D}^{CCS}}$, we find that its states are terms in a language containing CCS. So, we have maps $\mathrm{ob}(CCS) \overset{\theta}{\hookrightarrow} \mathrm{ob}(\mathcal{T}_{\mathbb{D}^{CCS}}) \xrightarrow{[\![-]\!]} \mathrm{ob}(\mathcal{S}_{\mathbb{D}^{CCS}})$, where ob takes the set of vertices, and with $[\![-]\!] \circ \theta = (\!|-|\!)$. Now, a problem is that CCS and the other two are LTSs on different alphabets, respectively $\mathbb{A}$ and $\mathbb{F}_{\mathbb{D}^{CCS}}$. We thus define morphisms $\mathbb{A} \xleftarrow{\xi} \mathcal{L} \xrightarrow{\chi} \mathbb{F}_{\mathbb{D}^{CCS}}$ and obtain by successive change of base (pullback when rewinding an arrow, postcomposition when following one) a

strong bisimulation $[\![-]\!]\colon \mathcal{T}^{\mathbb{A}}_{\mathbb{D}^{CCS}} \to \mathcal{S}^{\mathbb{A}}_{\mathbb{D}^{CCS}}$ over $\mathbb{A}$. We then prove that θ, viewed as a map $\mathrm{ob}(CCS) \hookrightarrow \mathrm{ob}(\mathcal{T}^{\mathbb{A}}_{\mathbb{D}^{CCS}})$, is included in weak bisimilarity, which yields for all $P, P \simeq_{\mathbb{A}} (\!|P|\!)$ (Corollary 4.5). Finally, drawing inspiration from Rensink et al. [33], we prove that CCS and $\mathcal{S}^{\mathbb{A}}_{\mathbb{D}^{CCS}}$ both have enough $\mathbb{A}$-*trees*, in a suitable sense, and that this, together with Corollary 4.5, entails the main result.

Related Work. Trying to reconcile two mainstream approaches to denotational semantics, we have designed a (first version of a) general framework aiming at an effective theory of programming languages. Other such frameworks exist [31, 32, 36, 10, 4, 2, 17, 1], but most of them, with the notable exception of Kleene coalgebra, attempt to organise the traditional techniques of syntax with variable binding and reduction rules into some algebraic structure. Here, as in Kleene coalgebra, syntax and its associated LTS are derived notions. Our approach may thus be seen as an extension of Kleene coalgebra to an innocent/multi-player setting, yet ignoring quantitative aspects.

In another sense of the word 'framework', recent work of Winskel and colleagues [34] investigates a general notion of concurrent game, based on earlier work by Melliès [26]. In our approach, the idea is that each programming language is interpreted as a playground, and that morphisms of playgrounds denote translations between languages. Winskel et al., instead, construct a (large) bicategory, into which each programming language should embed. Beyond this crucial difference, both approaches use presheaves and factorisation systems, and contain a notion of innocent, concurrent strategy. The precise links between the original notion of innocence, theirs, and ours remain to be better investigated.

Melliès's work [27], although in a deterministic and linear setting, incorporates some 'concurrency' into plays by presenting them as string diagrams. Our innocentisation procedure further bears some similarity with Harmer et al.'s [14] presentation of innocence based on a distributive law. Hildebrandt's approach to fair testing equivalence [16] uses closely related techniques, e.g., presheaves and sheaves — indeed, our innocence condition may be viewed as a sheaf condition. However, (1) his model falls in the aforementioned category of presheaf models for which parallel composition is a complex operation; and (2) he uses sheaves to correctly incorporate infinite behaviour in the model, which is different from our notion of innocence. Finally, direct inspiration is drawn from Girard [12], one of whose aims is to bridge the gap between syntax and semantics.

Perspectives. We plan to adapt our semantics to more complicated calculi like π, the Join and Ambients calculi, functional calculi, possibly with extra features (e.g., references, data abstraction, encryption), with a view to eventually generalising it. Preliminary investigations already led to a playground for π, whose adequacy remains to be established. More speculative directions include (1) defining a notion of morphisms for playgrounds, which should induce translations between strategies, and find sufficient conditions for such morphisms to preserve, resp. reflect testing equivalences; (2) generalising playgrounds to apply them beyond programming language semantics; in particular, preliminary work

shows that playgrounds easily account for cellular automata; this raises the question of how morphisms of playgrounds would compare with existing notions of simulations between cellular automata [8]; (3) trying and recast the issue of deriving transition systems (LTSs) from reductions [35] in terms of playgrounds.

Notation. Set is the category of sets; set is a skeleton of the category of finite sets, namely the category of finite ordinals and arbitrary maps between them; ford is the category of finite ordinals and monotone maps between them. For any category $\mathbb{C}$, $\widehat{\mathbb{C}} = [\mathbb{C}^{op}, \mathsf{Set}]$ denotes the category of presheaves on $\mathbb{C}$, while $\widehat{\mathbb{C}}^f = [\mathbb{C}^{op}, \mathsf{set}]$ and $\widehat{\mathbb{C}} = [\mathbb{C}^{op}, \mathsf{ford}]$ respectively denote the categories of presheaves of finite sets and of finite ordinals. One should distinguish, e.g., 'presheaf of finite sets' $\mathbb{C}^{op} \to \mathsf{set}$ from 'finite presheaf of sets' $F\colon \mathbb{C}^{op} \to \mathsf{Set}$. The latter means that the disjoint union $\sum_{c \in \mathrm{ob}(\mathbb{C})} F(c)$ is finite. Throughout the paper, any finite ordinal n is seen as $\{1, \ldots, n\}$ (rather than $\{0, \ldots, n-1\}$).

The notion of LTS that we'll use here is a little more general than the usual one, but this does not change much. We thus refer to the long version for details. Let us just mention that we work in the category Gph of reflexive graphs, and that the category of LTSs over A is for us the slice category Gph/A. LTSs admit a standard change of base functor given by pullback, and its left adjoint given by postcomposition. Given any LTS $p\colon G \to A$, an edge in G is *silent* when it is mapped by p to an identity edge. This straightforwardly yields a notion of weak bisimilarity over A, which is denoted by $\simeq_A$.

Our (infinite) CCS terms are coinductively generated by the typed grammar

$$\frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P|Q} \qquad \frac{\Gamma, a \vdash P}{\Gamma \vdash \nu a.P} \qquad \frac{\ldots \qquad \Gamma \vdash P_i \qquad \ldots}{\Gamma \vdash \sum_{i \in n} \alpha_i.P_i}\ (n \in \mathbb{N})\,,$$

where α_i is either a, $\overline{a}$, for $a \in \Gamma$, or $\heartsuit$. The latter is a 'tick' move used in the definition of fair testing equivalence. As a syntactic facility, we here understand Γ as ranging over $\mathbb{N}$, i.e., the free names of a process always are $1 \ldots n$ for some n. E.g., Γ, a denotes just $n+1$, and $a \in \Gamma$ means $a \in \{1, \ldots, \Gamma\}$.

Definition 1.1. *Let $\mathbb{A}$ be the reflexive graph with vertices given by finite ordinals, edges $\Gamma \to \Gamma'$ given by $\varnothing$ if $\Gamma \neq \Gamma'$, and by $\Gamma + \Gamma + \{id, \heartsuit\}$ otherwise, $id\colon \Gamma \to \Gamma$ being the identity edge on Γ. Elements of the first summand are denoted by $a \in \Gamma$, while elements of the second summand are denoted by $\overline{a}$.*

We view terms as a graph *CCS* over $\mathbb{A}$ with the usual transition rules. The graph $\mathbb{A}$ only has 'endo'-edges; some LTSs below do use more general graphs.

2 Recalling the Game

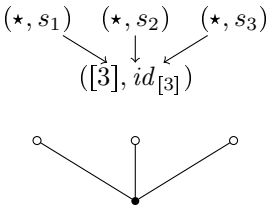
2.1 Positions, Moves, and Plays

In this section, we define plays in our game. For lack of space, we cannot be completely formal. A formal definition, with a gentle introduction to the required

techniques, may be found in HP (Section 3). Here is a condensed account. We start by defining a category $\mathbb{C}$. Then, *positions* in our game are defined to be particular finite presheaves in $\widehat{\mathbb{C}^f}$. *Moves* in our game are defined as certain *cospans* $X \xrightarrow{s} M \xleftarrow{t} Y$ in $\widehat{\mathbb{C}^f}$, where t indicates that Y is the *initial* position of the move, while s indicates that X is the *final* position. *Plays* are then defined as finite composites of moves in the bicategory $\mathsf{Cospan}(\widehat{\mathbb{C}^f})$ of cospans in $\widehat{\mathbb{C}^f}$. By construction, positions and plays form a subbicategory, called $\mathbb{D}_v^{CCS}$.

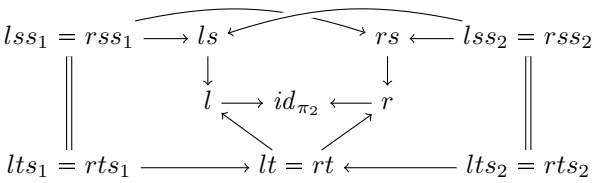
In order to motivate the definition of our base category $\mathbb{C}$, recall that (directed, multi) graphs may be seen as presheaves over the category freely generated by the graph with two objects $\star$ and $[1]$, and two edges $s, t \colon \star \to [1]$. Any presheaf G represents the graph with vertices in $G(\star)$ and edges in $G[1]$, the source and target of any $e \in G[1]$ being respectively $G(s)(e)$ and $G(t)(e)$. A way to visualise how such presheaves represent graphs is to compute their *categories of elements* [25]. Recall that the category of elements $\int G$ for a presheaf G over $\mathbb{C}$ has as objects pairs (c, x) with $c \in \mathbb{C}$ and $x \in F(c)$, and as morphisms $(c, x) \to (d, y)$ all morphisms $f \colon c \to d$ in $\mathbb{C}$ such that $F(f)(y) = x$. This category admits a canonical functor π_F to $\mathbb{C}$, and F is the colimit of the composite $\int F \xrightarrow{\pi_F} \mathbb{C} \xrightarrow{\mathsf{y}} \widehat{\mathbb{C}}$ with the Yoneda embedding. Hence, e.g., the category of elements for the representable presheaf over $[1]$ is the poset $(\star, s) \to ([1], id_{[1]}) \leftarrow (\star, t)$, which could be pictured as $\bullet \longrightarrow \bullet$, thus recovering some graphical intuition.

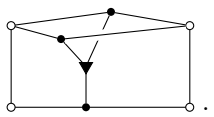
We now define our base category $\mathbb{C}$. Let us first give the raw definition, and then explain. $\mathbb{C}$ is freely generated from the graph $\mathbb{G}$, defined as follows, plus some equations. As objects, $\mathbb{G}$ has (1) an object $\star$, (2) an object $[n]$ for all $n \in \mathbb{N}$, (3) objects $o_{n,i}$ (output), $\iota_{n,i}$ (input), ν_n (channel creation), π_n^l (left fork), π_n^r (right fork), π_n (fork), $\heartsuit_n$ (tick), $\tau_{n,i,m,j}$ (synchronisation), for all $i \in n, j \in m, n, m \in \mathbb{N}$. $\mathbb{G}$ has edges, for all n, (1) $s_1^n, \ldots, s_n^n \colon \star \to [n]$, (2) $s^c, t^c \colon [n] \to c$, for all $c \in \{\pi_n^l, \pi_n^r, \heartsuit_n\} \cup (\cup_{i \in n} \{o_{n,i}, \iota_{n,i}\})$, (3) $[n+1] \xrightarrow{s^{\nu_n}} \nu_n \xleftarrow{t^{\nu_n}} [n]$, (4) $\pi_n^l \xrightarrow{l^n} \pi_n \xleftarrow{r^n} \pi_n^r$, $o_{n,i} \xrightarrow{\epsilon^{n,i,m,j}} \tau_{n,i,m,j} \xleftarrow{\rho^{n,i,m,j}} \iota_{m,j}$, for all $i \in n, j \in m$. In the following, we omit superscripts when clear from context. As equations, we require, for all n, m, $i \in n$, and $j \in m$, (1) $s^c \circ s_i^n = t^c \circ s_i^n$, (2) $s^{\nu_n} \circ s_i^{n+1} = t^{\nu_n} \circ s_i^n$, (3) $l \circ t = r \circ t$, (4) $\epsilon \circ t \circ s_i = \rho \circ t \circ s_j$.



In order to explain this seemingly arbitrary definition, let us compute a few categories of elements for representable presheaves. Let us start with an easy one, that of $[3]$ (we implicitly identify any $c \in \mathbb{C}$ with $\mathsf{y}c$). An easy computation shows that it is the poset pictured above. We will think of it as a position with one player $([3], id_{[3]})$ connected to three channels, and draw it as above, where the bullet represents the player, and circles represent channels. (The graphical representation is slightly ambiguous, but nevermind.) In particular, elements over $[3]$ represent ternary players, while elements over $\star$ represent channels. *Positions* are finite presheaves empty except perhaps on $\star$ and $[n]$'s. Let $\mathbb{D}_h^{CCS}$ be the subcategory of $\widehat{\mathbb{C}^f}$ consisting of positions and monic arrows between them.

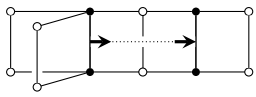
A more difficult category of elements is that of π_2. It is the poset generated by the graph on the left:

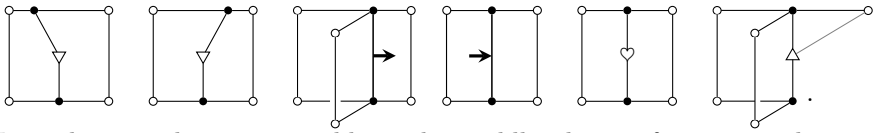




We think of it as a binary player (lt) forking into two players (ls and rs), and draw it as on the right. The vertical edges on the outside are actually identities: the reason we draw separate vertices is to identify the top and bottom parts of the picture as the respective images of both legs of the following cospan. First, consider the inclusion $[2] \mid [2] \hookrightarrow \pi_2$: its domain is any pushout of $[s_1, s_2]$: $(\star + \star) \to [2]$ with itself, i.e., the position consisting of two binary players sharing their channels; and the inclusion maps it to the top part of the picture. Similarly, we have a map $[2] \hookrightarrow \pi_2$ given by the player lt and its channels (the bottom part). The cospan $[2] \mid [2] \to \pi_2 \leftarrow [2]$ is called the *local fork move* of arity 2.

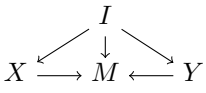
For lack of space, we cannot spell out all such categories of elements and cospans. We give pictorial descriptions for $(m, j, n, i) = (3, 3, 2, 1)$ of $\tau_{m,j,n,i}$ on the right and of π^l_n, π^r_n, $o_{m,j}$, $\iota_{n,i}$, $\heartsuit_n$, and ν_n below:





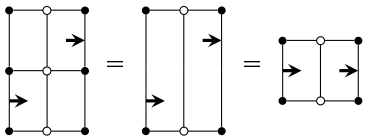
In each case, the representable is the middle object of a cospan determined by the top and bottom parts of the picture. E.g., for synchronisation we have $[m]_{\,j}|_i\,[n] \xrightarrow{s} \tau_{m,j,n,i} \xleftarrow{t} [m]_{\,j}|_i\,[n]$, where $[m]_{\,j}|_i\,[n]$ denotes the position X with one m-ary player x, one n-ary player y, such that $X(s_j)(x) = X(s_i)(y)$. Note that there is a crucial design choice in defining the legs of these cospans, which amounts to choosing initial and final positions for our moves.

These cospans altogether form the set of *local moves*, and are the 'seeds' for (global) moves, in the following sense. Calling an *interface* any presheaf consisting only of channels, local moves may be equipped with a canonical interface, consisting of the channels of their initial position. If $X \xrightarrow{s} M \xleftarrow{t} Y$ is a local move (with final position X), and I is its canonical interface, we obtain a commuting diagram (1) in $\widehat{\mathbb{C}^f}$ (with all arrows monic). For any morphism $I \to Z$ to some position Z, pushing $I \to X$, $I \to M$, and $I \to Y$ along $I \to Z$ yields, by universal property of pushout, a new cospan, say $X' \to M' \leftarrow Y'$. Letting *(global) moves* be all cospans obtained in this way, and plays be all composites of moves in $\mathsf{Cospan}(\widehat{\mathbb{C}^f})$, we obtain, as promised a subbicategory $\mathbb{D}^{CCS}_v$.



(1)

Passing from local to global moves allows moves to occur in larger positions. Furthermore, we observe that plays feature some concurrency. For instance, composing two global moves as on the right, we obtain a play in which the order of appearance of moves is no longer visible. In passing, this play embeds into a synchronisation, but is not one, since the input and output moves are not related. This play may be understood as each player communicating with the outside world. We conclude with a useful classification of moves.


(2)

Definition 2.1. *A move is* full *iff it is neither a left nor a right fork. We call* $\mathbb{F}$ *the graph of global, full moves.*

Intuitively, a move is full when its final position contains all possible avatars of involved players.

3 Behaviours, Strategies, and Fair Testing

3.1 Behaviours

Recall from HP the category $\mathbb{E}$ whose objects are maps $U \leftarrow X$ in $\widehat{\mathbb{C}}$, such that there exists a play $Y \to U \leftarrow X$, i.e., objects are plays, where we forget the final position. Its morphisms $(U \leftarrow X) \to (U' \leftarrow X')$ are commuting diagrams as on the right with all arrows monic. Morphisms $U \to U'$ in $\mathbb{E}$ represent extensions of U, both spatially (i.e., embedding into a larger position) and dynamically (i.e., adding more moves).

$$\begin{array}{ccc} U & \longrightarrow & U' \\ \uparrow & & \uparrow \\ X & \longrightarrow & X' \end{array}$$

We may relativise this category $\mathbb{E}$ to a particular position X, yielding a category $\mathbb{E}(X)$ of plays on X: take the fibre over X of the functor $\mathrm{cod}\colon \mathbb{E} \to \mathbb{D}_h^{CCS}$ mapping any play $U \leftarrow X$ to its initial position X. The objects of $\mathbb{E}(X)$ are just plays $(U \leftarrow X)$ on X, and morphisms are morphisms of plays whose lower border is id_X. This leads to a category of 'naive' strategies, called behaviours.

Definition 3.1. *The category* B_X *of* behaviours *on* X *is the category* $\widehat{\mathbb{E}(X)^f}$ *of presheaves of finite sets on* $\mathbb{E}(X)$.

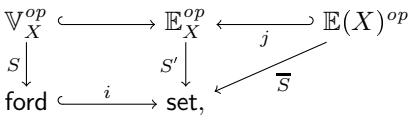
Behaviours suffer from the deficiency of allowing unwanted cooperation between players. HP (Example 12) exhibits a behaviour where players choose with whom they synchronise, which clearly is not allowed in CCS.

3.2 Strategies

To rectify this, we consider the full subcategory $\mathbb{V}$ of $\mathbb{E}$ consisting of *views*, i.e., compositions of basic local moves. We relativise views to a position X, as follows. Let, for any $n \in \mathbb{N}$, $[n]$ denote the single n-ary player, i.e., a single player connected to n distinct channels. Players of X are in 1-1 correspondence with

pairs (n, x), with $x\colon [n] \to X$ in $\mathbb{D}_h^{CCS}$. Relativisation of $\mathbb{V}$ to X is given by the category $\mathbb{V}_X$ with as objects all pairs (V, x), where $x\colon [n] \to X$, and V is a view with initial position $[n]$. Morphisms are induced by those of $\mathbb{E}$.

Definition 3.2. *The category* S_X *of* strategies *on* X *is the category* $\widehat{\mathbb{V}_X}$ *of presheaves of finite ordinals on* $\mathbb{V}_X$.



This rules out undesired behaviours. Recall from HP how to map strategies to behaviours: let first $\mathbb{E}_X$ be the category obtained as $\mathbb{V}_X$ from all plays instead of just views. Then, starting from a strategy S, let S' be obtained by right Kan extension of $i \circ S$ (by $\mathbb{V}_X^{op} \hookrightarrow \mathbb{E}_X^{op}$ being full and faithful), and let $\overline{S} = S' \circ j$. The assignment $S \mapsto \overline{S}$ extends to a full and faithful functor $\overline{(-)}\colon \mathsf{S}_X \to \mathsf{B}_X$. Furthermore, $\overline{(-)}$ admits a left adjoint, which we call *innocentisation*, maping naive strategies (behaviours) to innocent ones. By standard results [24], we have for any S: $\overline{S}(U) = \int_{v \in \mathbb{V}_X} S(v)^{\mathbb{E}_X(v,U)}$. Equivalently, $\overline{S}(U)$ is a limit of $(\mathbb{V}_X/U)^{op} \xrightarrow{\text{dom}} \mathbb{V}_X^{op} \xrightarrow{S} \mathsf{ford} \hookrightarrow \mathsf{set}$.

3.3 Decomposition: A Syntax for Strategies

Our definition of strategies is rather semantic in flavour. Indeed, presheaves are akin to domain theory. However, they also lend themselves well to a syntactic description. First, it is shown in HP that strategies on an arbitrary position X are in 1-1 correspondence with families of strategies indexed by the players of X. Recall that $[n]$ is the position consisting of one n-ary player, and that players of X may be defined as elements of $\mathrm{Pl}(X) = \sum_{n \in \mathbb{N}} \mathbb{D}_h^{CCS}([n], X)$.

Proposition 3.3. *We have* $\mathsf{S}_X \cong \prod_{(n,x) \in \mathrm{Pl}(X)} \mathsf{S}_{[n]}$. *For any* $S \in \mathsf{S}_X$, *we denote by* $S_{(n,x)}$ *the component corresponding to* $(n, x) \in \mathrm{Pl}(X)$ *under this isomorphism.*

This result yields a construction letting two strategies interact along an *interface*, i.e., a position consisting only of channels. This will be the basis of our semantic definition of fair testing equivalence. Consider any pushout Z of $X \leftarrow I \rightarrow Y$ where I is an interface. We have

Corollary 3.4. $\mathsf{S}_Z \cong \mathsf{S}_X \times \mathsf{S}_Y$.

Proof. We have $\mathbb{V}_Z \cong \mathbb{V}_X + \mathbb{V}_Y$, and conclude by universal property of coproduct.

We denote by $[S, T]$ the image of $(S, T) \in \mathsf{S}_X \times \mathsf{S}_Y$ under this isomorphism.

So, strategies over arbitrary positions may be decomposed into strategies over 'typical' players $[n]$. Let us now explain that strategies over such players may be further decomposed. For any strategy S on $[n]$ and basic move $b\colon [n'] \to [n]$, let the *residual* $S \cdot b$ of S after b be the strategy playing like S after b, i.e., for all $v \in \mathbb{V}_{[n']}$, $(S \cdot b)(v) = S(b \bullet v)$, where $\bullet$ denotes composition in $\mathbb{D}_v^{CCS}$. S is almost determined by its residuals. The only information missing from the $S \cdot b$'s to reconstruct S is the set of initial states and how they relate to the initial

states of each $(S \cdot b)$. Thus, for any position X, let id^v_X denote the identity play on X (i.e., nothing happens). For any initial state $\sigma \in S(id_{[n]})$, let $S_{|\sigma}$ be the restriction of S to states derived from σ, i.e., for all v, those $\sigma' \in S(v)$ which are mapped to σ under the restriction $S(!)\colon S(v) \to S(id_{[n]})$. S is determined by its set $S(id_{[n]})$ of initial states, plus the function $(\sigma, b) \mapsto (S_{|\sigma} \cdot b)$. Since $S(id_{[n]})$ is a finite ordinal m, we have for all n:

Theorem 3.5. $\mathsf{S}_{[n]} \cong \sum_{m\in\mathbb{N}}(\prod_{b\colon [n']\to[n]} \mathsf{S}_{[n']})^m \cong (\prod_{b\colon [n']\to[n]} \mathsf{S}_{[n']})^\star$.

This result may be understood as saying that strategies form a fixpoint of a certain (polynomial [23]) endofunctor of $\mathsf{Set}/\mathbb{I}$, where $\mathbb{I}$ is the set of 'typical' players $[n]$. This may be strengthened to show that they form a terminal coalgebra, i.e, that they are in bijection with infinite terms in the following typed grammar, with judgements $n \vdash_{\mathsf{D}} D$ and $n \vdash S$, where D is called a *definite prestrategy* and S is a *strategy*:

$$\frac{\ldots\ n_b \vdash S_b\ \ldots\ (\forall b\colon [n_b] \to [n] \in [\mathbb{B}]_n)}{n \vdash_{\mathsf{D}} \langle (S_b)_{b\in[\mathbb{B}]_n}\rangle} \qquad \frac{\ldots\ n \vdash_{\mathsf{D}} D_i\ \ldots\ (\forall i \in m)}{n \vdash \oplus_{i\in m} D_i}\ (m \in \mathbb{N}),$$

where $[\mathbb{B}]_n$ denotes the set of all isomorphism classes of basic moves from $[n]$. We need to use isomorphism classes here, because strategies may not distinguish between different, yet isomorphic basic moves. This achieves the promised syntactic description of strategies. We may readily define the translation of CCS processes, coinductively, as follows. For processes with channels in Γ, we define

$$\begin{array}{rlrl} (\!|\textstyle\sum_{i\in n} \alpha_i.P_i|\!) &= \langle b \mapsto \oplus_{\{i\in n \mid b=(\!|\alpha_i|\!)\}} (\!|P_i|\!)\rangle & (\!|a|\!) &= \iota_{\Gamma,a} \\ (\!|\nu a.P|\!) &= \langle \nu_\Gamma \mapsto (\!|P|\!), \_ \mapsto \varnothing\rangle & (\!|\overline{a}|\!) &= o_{\Gamma,a} \\ (\!|P \mid Q|\!) &= \langle \pi^l_\Gamma \mapsto (\!|P|\!), \pi^r_\Gamma \mapsto (\!|Q|\!), \_ \mapsto \varnothing\rangle & (\!|\heartsuit|\!) &= \heartsuit_\Gamma. \end{array}$$

E.g., $a.P + a.Q + \bar{b}.R$ is mapped to $\langle \iota_{\Gamma,a} \mapsto ((\!|P|\!) \oplus (\!|Q|\!)), o_{\Gamma,b} \mapsto (\!|R|\!), \_ \mapsto \varnothing\rangle$.

3.4 Semantic Fair Testing

We may now recall our semantic analogue of fair testing equivalence.

Definition 3.6. Closed-world *moves are (the global variants of)* ν,$\heartsuit$,π_n, *and* $\tau_{n,i,m,j}$. *A play is* closed-world *when it is a composite of closed-world moves.*

Let a closed-world play be *successful* when it contains a $\heartsuit$ move. Let then $\perp\!\!\!\perp_Z$ denote the set of behaviours B such that for any unsuccessful, closed-world play $U \leftarrow Z$ and $\sigma \in B(U)$, there exists $f\colon U \to U'$, with U' closed-world and successful, and $\sigma' \in B(U')$ such that $B(f)(\sigma') = \sigma$. Finally, let us say that a triple (I, h, S), for any $h\colon I \to X$ and strategy $S \in \mathsf{S}_X$, *passes* the test consisting of a morphism $k\colon I \to Y$ of positions and a strategy $T \in \mathsf{S}_Y$ iff $\overline{[S,T]} \in \perp\!\!\!\perp_Z$, where Z is the pushout of h and k. Let $S^{\perp\!\!\!\perp}$ denote the set of all such (k, T).

Definition 3.7. *For any* $h\colon I \to X$, $h'\colon I \to X'$, $S \in \mathsf{S}_X$, *and* $S' \in \mathsf{S}_{X'}$, $(I, h, S) \sim_f (I, h', S')$ *iff* $(I, h, S)^{\perp\!\!\!\perp} = (I, h', S')^{\perp\!\!\!\perp}$.

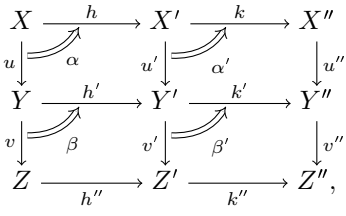
This yields an equivalence relation, analogous to standard fair testing equivalence, which we hence also call fair testing equivalence.

We have defined a translation $(\!|-|\!)$ of CCS processes to strategies, which raises the question of whether it preserves or reflects fair testing equivalence. The rest of the paper is devoted to proving that it does both.

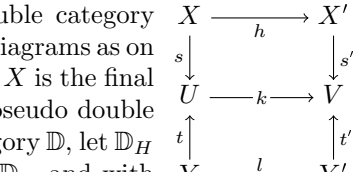
4 Playgrounds and Main Result

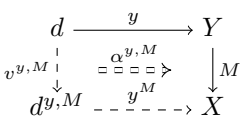
4.1 Playgrounds: A Theory of Individuality and Atomicity

We start by trying to give an idea of the notion of playground. To start with, we organise the game into a *(pseudo) double category* [13, 11]. This is a weakening of Ehresmann's double categories [9], where one direction has non strictly associative composition. Although we consider proper pseudo double categories, we often may treat them safely as double categories. A pseudo double category $\mathbb{D}$ consists of a set $\mathrm{ob}(\mathbb{D})$ of *objects*, shared by two categories $\mathbb{D}_h$ and $\mathbb{D}_v$. $\mathbb{D}_h$ is called the *horizontal* category of $\mathbb{D}$, and $\mathbb{D}_v$ is the *vertical* category. Composition in $\mathbb{D}_h$ is denoted by $\circ$, while we use $\bullet$ for $\mathbb{D}_v$. $\mathbb{D}$ is furthermore equipped with a set of *double cells* α, which have vertical, resp. horizontal, domain and codomain, denoted by $\mathrm{dom}_v(\alpha)$, $\mathrm{cod}_v(\alpha)$, $\mathrm{dom}_h(\alpha)$, and $\mathrm{cod}_h(\alpha)$. We picture this as, e.g., α above, where $u = \mathrm{dom}_h(\alpha)$, $u' = \mathrm{cod}_h(\alpha)$, $h = \mathrm{dom}_v(\alpha)$, and $h' = \mathrm{cod}_v(\alpha)$. $\mathbb{D}$ is furthermore equipped with operations for composing double cells: $\circ$ composes them along a common vertical morphism, $\bullet$ composes along horizontal morphisms. Both vertical compositions (of morphisms and double cells) may only be associative up to coherent isomorphism. The full axiomatisation is given by Garner [11], and we here only mention the *interchange law*, which says that the two ways of parsing the above diagram coincide: $(\beta' \circ \beta) \bullet (\alpha' \circ \alpha) = (\beta' \bullet \alpha') \circ (\beta \bullet \alpha)$.

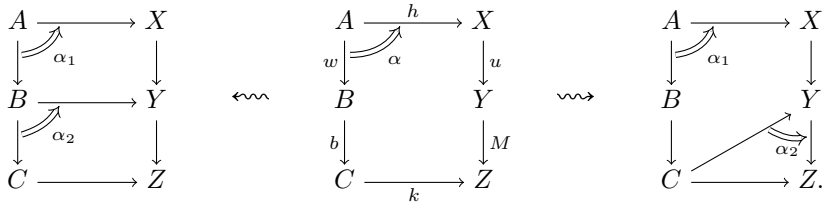


Example 4.1. Returning to the game, we have seen that positions are the objects of the category $\mathbb{D}_h^{CCS}$, whose morphisms are embeddings of positions. But positions are also the objects of the bicategory $\mathbb{D}_v^{CCS}$, whose morphisms are plays. It should seem natural to define a pseudo double category structure with double cells given by commuting diagrams as on the right in $\widehat{\mathbb{C}}$. Here, Y is the initial position and X is the final one; all arrows are mono. This indeed forms a pseudo double category $\mathbb{D}^{CCS}$. Furthermore, for any double category $\mathbb{D}$, let $\mathbb{D}_H$ be the category with objects all morphisms of $\mathbb{D}_v$, and with morphisms $u \to u'$ all double cells α such that $\mathrm{dom}_h(\alpha) = u$ and $\mathrm{cod}_h(\alpha) = u'$. A crucial feature of $\mathbb{D}^{CCS}$ is that the canonical functor $\mathrm{cod}_v \colon \mathbb{D}_H \to \mathbb{D}_h$ mapping any such α to $\mathrm{cod}_v(\alpha)$ is a Grothendieck fibration [21]. This means that one may canonically 'restrict' a play, say $u' \colon X' \to Y'$, along a horizontal morphism $h' \colon Y \to Y'$, and obtain a universal cell as α above, in a suitable sense.





Playgrounds are pseudo double categories with extra data and axioms, the first of which is that cod_v should be a fibration. To give a brief idea of further axioms, a playground $\mathbb{D}$ is equipped with a set of objects $\mathbb{I}$, called *individuals*, which correspond to our 'typical' players above. Let $\mathrm{Pl}(X) = \sum_{d \in \mathbb{I}} \mathbb{D}_h(d, X)$ denote the set of players of X. It also comes with classes $\mathbb{F}$ and $\mathbb{B}$ of *full*, resp. *basic* moves; and every play (i.e., vertical morphism) is assumed to admit a decomposition into moves in $\mathbb{F} \cup \mathbb{B}$ (hence *atomicity*). Basic moves are assumed to have individuals as both domain and codomain, and *views* are defined to be composites of basic moves. The crucial axiom for innocence to behave well assumes that, for any position Y and player $y \colon d \to Y$, there exists a cell $\alpha^{y,M}$ as above, with $v^{y,M}$ a view, which is unique up to canonical isomorphism of such. Intuitively: any player in the final position of a play has an essentially unique view of the play. A last, sample axiom shows how some sequentiality is enforced, which is useful to tame the concurrency observed in (2). It says that any double cell as in the center below, where b is a basic move and M is any move, decomposes in exactly one of the forms on the left and right:



The idea is that, C being an individual, if M has a non-trivial restriction to C, then b must be one of its views. Again, for the formal definition, see [18].

Proposition 4.2. $\mathbb{D}^{CCS}$ *forms a playground (basic moves being the* local *ones).*

4.2 Syntaxes and Labelled Transition Systems

Notions of residuals and restrictions defined above for CCS are easily generalised to arbitrary playgrounds. They lead to the exact same syntax as in the concrete case (below Theorem 3.5). They further yield a first, naive LTS over full moves for strategies. The intuition is that there is a transition $S \xrightarrow{M} S'$, for any full move M, when $S \cdot M = S'$. (Residuals $S \cdot M$ are here defined analogously to the case of basic moves $S \cdot b$ above.) An issue with this LTS is that $S \cdot M$ may have several possible initial states, and we have seen that it makes more sense to restrict to a single state before taking residuals. We thus define our LTS $\mathcal{S}_{\mathbb{D}}$ to have as vertices pairs (X, S) of a position X and a *definite* strategy S, i.e., a strategy with exactly one initial state (formally, $S_{(d,x)}(id_d) = 1$ for all $(d, x) \in \mathrm{Pl}(X)$ — recalling that id_d is an (initial) object in $\mathbb{V}_d$). We then say that there is a transition $(X, S) \xrightarrow{M} (X', S')$ for any full move $M \colon X' \to X$, when $S' = (S \cdot M)_{|\sigma'}$, for some initial state σ' of $S \cdot M$.

Example 4.3. Consider a strategy of the shape $S = \langle \pi_n^r \mapsto S_1, \pi_n^l \mapsto S_2, \_ \mapsto \varnothing \rangle$ on $[n]$, with definite S_1 and S_2. There is a π_n transition to the position with two n-ary players x_1 and x_2, equipped with S_1 and S_2, respectively. If now S_1 and S_2 are not definite, any π_n transition has to pick initial states $\sigma_1 \in S_1(id_{[n]})$ and $\sigma_2 \in S_2(id_{[n]})$, i.e., $S \xrightarrow{\pi_n} [(S_1)_{|\sigma_1}] \mid [(S_2)_{|\sigma_2}]$. Here, we use a shorthand notation for pairs (X, S), defined as follows. First, for any strategy S over $[n]$ and position X with exactly one n-ary player x and names in Γ, we denote by $\Gamma \vdash [x : S](a_1, \ldots, a_n)$ the pair (X, S), where $a_i = X(s_i)(x)$, for all $i \in n$. If now X has several players, say $x_1, \ldots, x_p$, of respective arities $n_1, \ldots, n_p$, and $S_1, \ldots, S_p$ are strategies of such arities, we denote by $\Gamma \vdash [x_1 : S_1](a_1^1, \ldots, a_{n_1}^1) \mid \ldots \mid [x_p : S_p](a_1^p, \ldots, a_{n_p}^p)$ the pair $(X, [S_1, \ldots, S_p])$. When they are irrelevant, we often omit Γ, the x_j's, and the a_i^j's, as in our example.

Beyond the one for strategies, there is another syntax one can derive from any playground. Instead of relying on basic moves as before, one now relies on full moves. Thinking of full moves as inference rules (e.g., in natural deduction), the premises of the rule for any full $M \colon X \to Y$ should be those players (d_x, x) of X whose view through M is non-trivial, i.e., is a basic move. We call this set of players $\mathrm{Pl}(M)$. The natural syntax rule is thus the first one above (glossing over some details), which defines *process terms* T. We add a further rule for guarded sum allowing to choose between several moves. One has to be a little careful here, and only allow moves $M \colon X \to Y$ such that $\mathrm{Pl}(M)$ is a singleton. This yields the second rule above, where $n \in \mathbb{N}$, and $\forall i \in n$, M_i is such a move and d_i is the arity of the unique element of $\mathrm{Pl}(M_i)$. Calling T_d the set of infinite terms for this syntax, there is a natural translation map $[\![-]\!] \colon \mathsf{T}_d \to \mathsf{S}_d$ to strategies, for all $d \in \mathbb{I}$, which looks a lot like $(\!|-|\!)$, and an LTS $\mathcal{T}_{\mathbb{D}}$, whose vertices are pairs (X, T) of a position X, with $T \in \prod_{d,x \in \mathrm{Pl}(X)} \mathsf{T}_d$. The main result on playgrounds is

$$\frac{\ldots\ d_x \vdash T_x\ \ldots}{d \vdash M\langle (T_x)_{x \in \mathrm{Pl}(M)} \rangle} \qquad \frac{\ldots\ d_i \vdash T_i\ \ldots\ (\forall i \in n)}{d \vdash \sum_{i \in n} M_i.T_i}$$

Theorem 4.4. *The map $[\![-]\!] \colon \mathcal{T}_{\mathbb{D}} \to \mathcal{S}_{\mathbb{D}}$ is a functional, strong bisimulation.*

4.3 Change of Base and Main Result

The LTS $\mathcal{S}_{\mathbb{D}^{CCS}}$ obtained for $\mathbb{D}^{CCS}$ is much too fine to be relevant for bisimilarity to make behavioural sense. E.g., the translations of $a \,|\, b$ and $b \,|\, a$ are not bisimilar. Indeed, labels, i.e., full moves in $\mathbb{F}_{\mathbb{D}^{CCS}}$, bear the information of which player is involved in the transition. So both strategies have a π_Γ translation to a position with two Γ-ary players, say x_1 and x_2. But then, $a \mid b$ has a transition where x_1 plays an input on a, which $b \mid a$ cannot match. Refining the above notation, and omitting $(\!|-|\!)$, we may write the former transitions as $[a \mid b] \xrightarrow{\pi_\Gamma} [a] \mid [b] \xrightarrow{x_1, \iota_{\Gamma,a}} [0] \mid [b]$. There is another problem with this LTS, namely that there are undue transitions. E.g., we have $[\nu a.a] \xrightarrow{\nu_0} [a] \xrightarrow{\iota(a),a} 0$. The transition system does not yet take privacy of channels into account.

Let us first rectify the latter deficiency. To this end, we pull back our LTS $\mathcal{S}_{\mathbb{D}^{CCS}} \to \mathbb{F}_{\mathbb{D}^{CCS}}$ along a morphism of graphs $\mathcal{L} \to \mathbb{F}_{\mathbb{D}^{CCS}}$ defined as follows. Let $\mathcal{L}$

have *interfaced positions* as vertices, i.e., morphisms $h\colon I \to X$ from an interface to a position. I specifies the public channels, and hence we let edges $h \to h'$ be commuting diagrams of the shape (1), where M may be any full move (X being the final position), except inputs and outputs on a channel outside the image of I. We then straightforwardly define $\chi\colon \mathcal{L} \to \mathbb{F}_{\mathbb{D}^{CCS}}$ to map h to X and any diagram above to M. The pullback $\mathcal{S}^{\mathcal{L}}_{\mathbb{D}^{CCS}} \to \mathcal{L}$ of $\mathcal{S}_{\mathbb{D}^{CCS}}$ along χ is rid of undue communications on private channels.

To rectify the other deficiency mentioned above, recalling from Definition 1.1 that $\mathbb{A}$ is the alphabet for CCS, we define a morphism $\xi\colon \mathcal{L} \to \mathbb{A}$ by mapping $(I \to X)$ to its set $I(\star)$ of channels, and any M to (1) $\heartsuit$ if M is a tick move, (2) id if M is a synchronisation, a fork, or a channel creation, (3) a if M is an input on $a \in I(\star)$, (4) $\overline{a}$ if M is an output on $a \in I(\star)$. (Positions are formally defined as presheaves to set, hence channels directly form a finite ordinal number.) It is here crucial to have restricted attention to $\mathcal{L}$ beforehand, otherwise we would not know what to do with communications on private channels. Let $\mathcal{S}^{\mathbb{A}}_{\mathbb{D}^{CCS}} = \xi_!(\mathcal{S}^{\mathcal{L}}_{\mathbb{D}^{CCS}})$ be the post-composition of $\mathcal{S}^{\mathcal{L}}_{\mathbb{D}^{CCS}} \to \mathcal{L}$ with ξ.

The obtained LTS $\mathcal{S}^{\mathbb{A}}_{\mathbb{D}^{CCS}} \to \mathbb{A}$ is now ready for our purposes. Proceeding similarly for the LTS $\mathcal{T}_{\mathbb{D}^{CCS}}$ of process terms, we obtain a strong, functional bisimulation $[\![-]\!]\colon \mathrm{ob}(\mathcal{T}^{\mathbb{A}}_{\mathbb{D}^{CCS}}) \to \mathrm{ob}(\mathcal{S}^{\mathbb{A}}_{\mathbb{D}^{CCS}})$ over $\mathbb{A}$. We then prove that $\theta\colon \mathrm{ob}(CCS) \hookrightarrow \mathrm{ob}(\mathcal{T}^{\mathbb{A}}_{\mathbb{D}^{CCS}})$ is included in weak bisimilarity over $\mathbb{A}$, and, easily, that $(\!|-|\!) = [\![-]\!] \circ \theta$.

Corollary 4.5. *For all P, $P \simeq_{\mathbb{A}} (\!|P|\!)$.*

Furthermore, we prove that $\sim_f$ coincides with the standard, LTS-based definition of fair testing, i.e., $P \sim_{f,s} Q$ iff for all sensible T, $(P \mid T \in \bot_s) \Leftrightarrow (Q \mid T \in \bot_s)$, where $P \in \bot_s$ iff any $\heartsuit$-free reduction sequence $P \Rightarrow P'$ extends to one with $\heartsuit$. To obtain our main result, we finally generalise an observation of Rensink and Vogler [33], which essentially says that for fair testing equivalence in CCS, it is sufficient to consider a certain class of tree-like tests, called *failures*. We first slightly generalise the abstract setting of De Nicola and Hennessy [7] for testing equivalences, e.g., to accomodate the fact that strategies are indexed over interfaces. This yields a notion of *effective graph*. We then show that, for any effective graph G over an alphabet A, the result on failures goes through, provided G *has enough A-trees*, in the sense that, up to mild conditions, for any tree t over A, there exists $x \in G$ such that $x \simeq_A t$. Consequently, for any relation $R\colon G \nrightarrow G'$ between two such effective graphs with enough A-trees, if R is included in weak bisimilarity over A, then R preserves and reflects fair testing equivalence. We thus obtain our main result:

Theorem 4.6. *For any $\Gamma \in \mathbb{N}$, let I_Γ be the interface consisting of Γ channels, and $h_\Gamma\colon I_\Gamma \to [\Gamma]$ be the canonical inclusion. For any CCS processes P and Q over Γ, we have $P \sim_{f,s} Q$ iff $(I_\Gamma, h_\Gamma, (\!|P|\!)) \sim_f (I_\Gamma, h_\Gamma, (\!|Q|\!))$.*

Remark 4.7. Until now, we have considered arbitrary, infinite CCS processes. Let us now restrict ourselves to recursive processes (e.g., in the sense of HP). We obviously still have that $(\!|P|\!) \sim_f (\!|Q|\!)$ implies $P \sim_{f,s} Q$. The converse is

less obvious and may be stated in very simple terms: suppose you have two recursive CCS processes P and Q and a test process T, possibly non-recursive, distinguishing P from Q; is there any recursive T' also distinguishing P from Q?

References

[1] Ahrens, B.: Initiality for typed syntax and semantics. In: Ong, L., de Queiroz, R. (eds.) WoLLIC 2012. LNCS, vol. 7456, pp. 127–141. Springer, Heidelberg (2012)

[2] Bonsangue, M., Rutten, J., Silva, A.: A kleene theorem for polynomial coalgebras. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 122–136. Springer, Heidelberg (2009)

[3] Brinksma, E., Rensink, A., Vogler, W.: Fair testing. In: Lee, I., Smolka, S.A. (eds.) CONCUR 1995. LNCS, vol. 962, pp. 313–327. Springer, Heidelberg (1995)

[4] Bruni, R., Montanari, U.: Cartesian closed double categories, their lambda-notation, and the pi-calculus. In: LICS. IEEE Computer Society (1999)

[5] Cacciagrano, D., Corradini, F., Palamidessi, C.: Explicit fairness in testing semantics. Logical Methods in Computer Science 5(2) (2009)

[6] Cattani, G.L., Winskel, G.: Presheaf models for concurrency. In: van Dalen, D., Bezem, M. (eds.) CSL 1996. LNCS, vol. 1258, pp. 58–75. Springer, Heidelberg (1996)

[7] De Nicola, R., Hennessy, M.: Testing equivalences for processes. Theor. Comput. Sci. 34 (1984)

[8] Delorme, M., Mazoyer, J., Ollinger, N., Theyssier, G.: Bulking I: An abstract theory of bulking. Theoretical Computer Science 412(30) (2011)

[9] Ehresmann, C.: Catégories et structures. Dunod (1965)

[10] Gadducci, F., Montanari, U.: The tile model. In: Plotkin, G.D., Stirling, C., Tofte, M. (eds.) Proof, Language, and Interaction. The MIT Press (2000)

[11] Garner, R.: Polycategories. PhD thesis, University of Cambridge (2006)

[12] Girard, J.-Y.: Locus solum: From the rules of logic to the logic of rules. Mathematical Structures in Computer Science 11(3) (2001)

[13] Grandis, M., Pare, R.: Limits in double categories. Cahiers de Topologie et Géométrie Différentielle Catégoriques 40(3) (1999)

[14] Harmer, R., Hyland, M., Melliès, P.-A.: Categorical combinatorics for innocent strategies. In: LICS. IEEE Computer Society (2007)

[15] Harmer, R., McCusker, G.: A fully abstract game semantics for finite nondeterminism. In: LICS. IEEE Computer Society (1999)

[16] Hildebrandt, T.T.: Towards categorical models for fairness: fully abstract presheaf semantics of SCCS with finite delay. Theoretical Computer Science 294(1/2) (2003)

[17] Hirschowitz, T.: Cartesian closed 2-categories and permutation equivalence in higher-order rewriting. Preprint (2010)

[18] Hirschowitz, T.: Full abstraction for fair testing in CCS (2012), Draft available from `http://lama.univ-savoie.fr/~hirschowitz`

[19] Hirschowitz, T., Pous, D.: Innocent strategies as presheaves and interactive equivalences for CCS. Scientific Annals of Computer Science 22(1) (2012), Selected papers from ICE 2011

[20] Hyland, J.M.E., Ong, C.-H.L.: On full abstraction for PCF: I, II, and III. Inf. Comput. 163(2) (2000)

[21] Jacobs, B.: Categorical Logic and Type Theory. Studies in Logic and the Foundations of Mathematics, vol. 141. North Holland, Amsterdam (1999)
[22] Joyal, A., Nielsen, M., Winskel, G.: Bisimulation and open maps. In: LICS. IEEE Computer Society (1993)
[23] Kock, J.: Polynomial functors and trees. International Mathematics Research Notices 2011(3) (2011)
[24] MacLane, S.: Categories for the Working Mathematician, 2nd edn. Graduate Texts in Mathematics, vol. 5. Springer, Heidelberg (1998)
[25] MacLane, S., Moerdijk, I.: Sheaves in Geometry and Logic: A First Introduction to Topos Theory. Universitext. Springer, Heidelberg (1992)
[26] Melliès, P.-A.: Asynchronous games 2: The true concurrency of innocence. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 448–465. Springer, Heidelberg (2004)
[27] Melliès, P.-A.: Game semantics in string diagrams. In: LICS. IEEE Computer Society (2012)
[28] Milner, R.: A Calculus of Communication Systems. LNCS, vol. 92. Springer, Heidelberg (1980)
[29] Milner, R.: Communication and Concurrency. Prentice-Hall (1989)
[30] Natarajan, V., Cleaveland, R.: Divergence and fair testing. In: Fülöp, Z. (ed.) ICALP 1995. LNCS, vol. 944, pp. 648–659. Springer, Heidelberg (1995)
[31] Nipkow, T.: Higher-order critical pairs. In: LICS. IEEE Computer Society (1991)
[32] Plotkin, G.D.: A structural approach to operational semantics. DAIMI Report FN-19, Computer Science Department, Aarhus University (1981)
[33] Rensink, A., Vogler, W.: Fair testing. Inf. Comput. 205(2) (2007)
[34] Rideau, S., Winskel, G.: Concurrent strategies. In: LICS. IEEE Computer Society (2011)
[35] Sewell, P.: From rewrite rules to bisimulation congruences. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 269–284. Springer, Heidelberg (1998)
[36] Turi, D., Plotkin, G.D.: Towards a mathematical operational semantics. In: LICS. IEEE Computer Society (1997)

A Simple Case of Rationality of Escalation

Pierre Lescanne

University of Lyon, École normale supérieure de Lyon, CNRS (LIP),
46 allée d'Italie, 69364 Lyon, France

Abstract. Escalation is the fact that in a game (for instance an auction), the agents play forever. It is not necessary to consider complex examples to establish its rationality. In particular, the $0,1$-game is an extremely simple infinite game in which escalation arises naturally and rationally. In some sense, it can be considered as the paradigm of escalation. Through an example of economic games, we show the benefit economics can take of coinduction.

Keywords: economic game, infinite game, sequential game, crash, escalation, speculative bubble, coinduction, auction.

> *[T]he future of economics is increasingly technical work that is founded on the vision that the economy is a complex system.*
>
> David Collander [6]

Sequential games are the natural framework for decision processes. In this paper we study a decision phenomenon called *escalation*. Finite sequential games (also known as extensive games) have been introduced by Kuhn [9] and subgame perfect equilibria have been introduced by Selten [19] whereas escalation has been introduced by Shubik [20]. Sequential games are games in which each player plays one after the other (or possibly after herself). In some specific infinite games, it has been showed that escalation cannot occur among rational players. Here we show on a simple example, the 0,1 game, that this is not the case if one uses coinduction. In addition the 0,1 game has nice properties which make it an excellent paradigm of escalation and a good domain of application for coalgebras and coinduction.

1 The Problem of Escalation

> *That "rational agents" should* not *engage in such [escalation] behavior seems obvious.*
>
> Wolfgang Leininger [11]

Escalation in sequential games is a classic of game theory and it is admitted that escalation is irrational. The rationality which we consider is that given

R. Heckel and S. Milius (Eds.): CALCO 2013, LNCS 8089, pp. 191–204, 2013.

by equilibria. It has been proved that in finite sequential games, rationality is obtained by a specific equilibrium called *backward induction* (see Appendix). More precisely a consequence of Aumann's theorem [2] says that an agent takes a rational decision in a finite sequential game if she makes her choice according to backward induction. In this paper we generalize backward induction into *subgame perfect equilibria* and we consider naturally that rationality is reached by subgame perfect equilibria (SPE in short) relying on Capretta's [5] extension of Aumann's theorem.

What is Escalation? In a sequential game, escalation is the possibility that agents take rational decisions forever without stopping. This phenomenon has been evidenced by Shubik [20] in a game called the *dollar auction*. Without being very difficult, its analysis is relatively involved, because it requires infinitely many strategy profiles indexed by $n \in \mathbb{N}$ [12]. Moreover in each step there are two and only two equilibria. By an observation of the past decisions of her opponent an agent could get a clue of her strategy and might this way avoid escalation. This blindness of the agents is perhaps not completely realistic and was criticized (see [14] Section 4.2). In this paper, we propose an example which is much simpler theoretically and which offers infinitely many equilibria at each step. Due to the form of the equilibria, the agent has no clue on which strategy is taken by her opponent.

Escalation and Infinite Games. Books and articles [7,8,17,11,16] which cover escalation take for granted that escalation is irrational. Following Shubik, all accept that escalation takes place and can only take place in an infinite game, but their argument uses a reasoning on finite games. Indeed, if one cuts the infinite game in which escalation is supposed to take place at a finite position, one gets a finite game, in which the only right and rational decision is to never start the game, because the only backward induction equilibrium corresponds to not start playing. Then the result is extrapolated by the authors to infinite games by making the size of the game to grow to infinity. However, it has been known for a long time at least since Weierstraß [24], that the "cut and extrapolate" method is wrong (see Appendix). For Weierstraß this would lead to the conclusion that the infinite sum of differentiable functions would be differentiable whereas he has exhibited a famous counterexample. In the case of infinite structures like infinite games, the right reasoning is coinduction. With coinduction we were able to show that the dollar auction has a rational escalation [15,14]. Currently, since the tools used generally in economics are pre-coinduction based, they conclude that bubbles and crises are impossible and everybody's experience has witnessed the opposite. Careful analysis done by quantitative economists, like for instance Bouchaud [3,4], have shown that bursts, which share much similarities with escalation, actually take place at any time scale. Escalation is therefore an intrinsic feature of economics. Consequently, coinduction is the tool that economists who call for a refoundation of economics [6,3] are waiting for [25].

Structure of the Paper. This paper is structured as follows. In Section 2 we present infinite games, infinite strategy profiles and infinite strategies, then we describe the 0,1-game in Section 3. Last, we introduce the concept of equilibrium (Sections 4 and 5) and we discuss escalation (Section 6). In an appendix, we talk about finite games and finite strategy profiles.

2 Two Choice Sequential Games

Our aim is not to present a general theory. For this the reader is invited to look at [1,14,15]. But we want to give a taste of infinite sequential games through a very simple one. This game has two agents and two choices. To support our claim about the rationality of escalation, we do not need more features. In [14], we have shown the existence of a big conceptual gap between finite games and infinite games.

Assume that the set P of agents is made of two agents called A and B. In this framework, an infinite sequential two choice game has two shapes. In the first shape, it is an ending position in which case it boils down to the distribution of the payoffs to the agents. In other words the game is reduced to a function $f : A \mapsto f_{\mathsf{A}}, B \mapsto f_{\mathsf{B}}$ and we write it $\langle f \rangle$. In the second shape, it is a generic game with a set C made of two potential choices: d or r (d for *down* and r for *right)*. Since the game is potentially infinite, it may continue forever. Thus formally in this most general configuration a game can be seen as a triple:

$$g = \langle p, g_d, g_r \rangle.$$

where p is an agent and g_d and g_r are themselves games. The subgame g_d corresponds to the down choice, i.e., the choice corresponding to go *down* and the subgame g_r corresponds to the *right* choice, i.e., the choices corresponding to go to the right. In other words, we define a functor:

$$\langle\ \rangle : \mathsf{X} \quad \to \quad \mathsf{Payoff} \quad + \quad \mathsf{P} \times \mathsf{X} \times \mathsf{X}.$$

of which Game is the final coalgebra and where $\mathsf{P} = \{\mathsf{A}, \mathsf{B}\}$ and $\mathsf{Payoff} = \mathbb{R}^{\mathsf{P}}$.

2.1 Strategy Profiles

From a game, one can deduce *strategy profiles* (later we will also say simply *profiles*), which is obtained by adding a label, at each node, which is a choice made by the agent. A choice belong to the set $\{d, r\}$. In other words, a strategy profile is obtained from a game by adding, at each node, a new label, namely a choice. Therefore a strategy profile which does not correspond to an ending game is a quadruple:

$$s = \langle\!\langle p, c, s_d, s_r \rangle\!\rangle,$$

where p is an agent (A or B), c is choice (d or r), and, s_d and s_r are two strategy profiles. The strategy profile which corresponds to an ending position has no

choice, namely it is reduced to a function $\langle\!\langle f \rangle\!\rangle = \langle\!\langle A \mapsto f_{\mathsf{A}}, B \mapsto f_{\mathsf{B}} \rangle\!\rangle$. From a strategy profile, one can build a game by removing the choices:

$$\mathsf{game}(\langle\!\langle f \rangle\!\rangle) = \langle f \rangle$$
$$\mathsf{game}(\langle\!\langle p, c, s_d, s_r \rangle\!\rangle) = \langle p, \mathsf{game}(s_d), \mathsf{game}(s_r) \rangle$$

$\mathsf{game}(s)$ is the underlying game of the strategy profile s.

Given a strategy profile s, one can associate, by induction, a (partial) payoff function $\widehat{s}$, as follows:

$$\begin{array}{llcl} \textit{when} & s = \langle\!\langle f \rangle\!\rangle & \widehat{s} & = & f \\ \textit{when} & s = \langle\!\langle p, d, s_d, s_r \rangle\!\rangle & \widehat{s} & = & \widehat{s_d} \\ \textit{when} & s = \langle\!\langle p, r, s_d, s_r \rangle\!\rangle & \widehat{s} & = & \widehat{s_r} \end{array}$$

$\widehat{s}$ is not defined if its definition runs in an infinite process. For instance, if $s_{\mathsf{A},\infty}$ is the strategy profile defined in Section 6, $\widehat{s_{\mathsf{A},\infty}}$ is not defined. To ensure that we consider only strategy profiles where the payoff function is defined we restrict to strategy profiles that are called *convergent*, written $s \downarrow$ (or sometimes prefixed $\downarrow(s)$) and defined as the least predicate satisfying

$$s = \langle\!\langle f \rangle\!\rangle \quad \vee \quad s = \langle\!\langle p, d, s_d, s_r \rangle\!\rangle \Rightarrow s_d \downarrow \ \wedge \langle\!\langle p, r, s_d, s_r \rangle\!\rangle \Rightarrow s_r \downarrow .$$

Proposition 1. *If $s \downarrow$, then $\widehat{s}$ is defined.*

Proof. By induction. If $s = \langle\!\langle f \rangle\!\rangle$, then since $\widehat{s} = f$ and f is defined, $\widehat{s}$ is defined.

If $s = \langle\!\langle p, c, s_d, s_r \rangle\!\rangle$, there are two cases: $c = d$ or $c = r$. Let us look at $c = d$. If $c = d$, $\widehat{s_d}$ is defined by induction and since $\widehat{s} = \widehat{s_d}$, we conclude that $\widehat{s}$ is defined.

The case $c = r$ is similar.

As we will consider the payoff function also for subprofiles, we want the payoff function to be defined on subprofiles as well. Therefore we define a property stronger than convergence which we call *strong convergence*. We say that a strategy profile s is *strongly convergent* and we write it $s \Downarrow$ if it is the largest predicate fulfilling the following conditions.

- $\langle\!\langle p, c, s_d, s_r \rangle\!\rangle \Downarrow$ if
  - $\langle\!\langle p, c, s_d, s_r \rangle\!\rangle$ is convergent,
  - s_d is strongly convergent,
  - s_r is strongly convergent.
- $\langle\!\langle f \rangle\!\rangle$ is always strongly convergent

More formally:

$$s \Downarrow \quad \Leftrightarrow_{\textcircled{c}} \quad s = \langle\!\langle f \rangle\!\rangle \ \vee \ (s = \langle\!\langle p, c, s_d, s_r \rangle\!\rangle \wedge s \downarrow \wedge s_d \Downarrow \wedge s_r \Downarrow).$$

There is however a difference between the definitions of $\downarrow$ and $\Downarrow$. Wherever $s \downarrow$ is defined *by induction*[1], from the ending games to the game, $s \Downarrow$ is defined *by coinduction*[2].

Both concepts are based on the fixed-point theorem established by Tarski [21]. The definition of $\Downarrow$ is typical of infinite games and means that $\Downarrow$ is invariant along the infinite game. To make the difference clear between the definitions, we use the symbol $\Leftarrow\!\!\textcircled{i}\!\!\Rightarrow$ for inductive definitions and the symbol $\Leftarrow\!\!\textcircled{c}\!\!\Rightarrow$ for coinductive definitions. By the way, the definition of the function game is also coinductive.

We can define the notion of *subprofile*, written $\precsim$:

$$s' \precsim s \quad \Leftarrow\!\!\textcircled{i}\!\!\Rightarrow \quad s' \sim_s s \ \vee \ s = \langle\!\langle p, c, s_d, s_r \rangle\!\rangle \wedge (s' \precsim s_d \vee s' \precsim s_r),$$

where $\sim_s$ is the bisimilarity among profiles defined as the largest binary predicate $s' \sim_s s$ such that

$$s' = \langle\!\langle f \rangle\!\rangle = s \quad \vee \quad (s' = \langle\!\langle p, c, s'_d, s'_r \rangle\!\rangle \wedge s = \langle\!\langle p, c, s_d, s_r \rangle\!\rangle \wedge s'_d \sim_s s_d \wedge s'_r \sim_s s_r).$$

Notice that since we work with infinite objects, we may have $s \not\sim_s s'$ and $s \precsim s' \precsim s$. In other words, an infinite profile can be a strict subprofile of itself. This is the case for $s_{1,0,a}$ and $s_{1,0,b}$ in Section 4. If a profile is strongly convergent, then the payoffs associated with all its subprofiles are defined.

Proposition 2. *If $s_1 \Downarrow$ and if $s_2 \precsim s_1$, then $\widehat{s_2}$ is defined.*

2.2 The Always Modality

We notice that $\downarrow$ characterizes a profile by a property of the head node, we would say that this property is local. $\Downarrow$ is obtained by distributing the property along the game. In other words we transform the predicate $\downarrow$ and such a predicate transformer is called a *modality*. Here we are interested by the modality *always*, also written $\Box$.

Given a predicate Φ on strategy profiles, the predicate $\Box P$ is defined coinductively as follows:

$$\Box\Phi(s) \Leftarrow\!\!\textcircled{c}\!\!\Rightarrow \Phi(s) \wedge s = \langle\!\langle p, c, s_d, s_r \rangle\!\rangle \Rightarrow (\Box\Phi(s_d) \wedge \Box\Phi(s_r)).$$

The predicate "is strongly convergent" is the same as the predicate "is always convergent".

Proposition 3. $s \Downarrow \quad \Leftrightarrow \quad \Box\downarrow(s)$.

[1] Roughly speaking a definition by induction works from the basic elements, here the ending games, to constructed elements.

[2] Roughly speaking a definition by coinduction works on infinite objects, like an invariant.

2.3 Strategies

The coalgebra of *strategies*[3] is defined by the functor

$$[\![\ ]\!] : X \to \mathbb{R}^{\mathsf{P}} \ + \ (\mathsf{P} + \mathsf{Choice}) \times X \times X$$

where $\mathsf{Choice} = \{d, r\}$. A strategy of agent p is a game in which some occurrences of p are replaced by choices. A strategy is written $[\![f]\!]$ or $[\![x, s_1, s_2]\!]$. By replacing the choice made by agent p by the agent p herself, we can associate a game with a pair consisting of a strategy and an agent:

$$\begin{array}{lcl} \mathsf{st2g}([\![f]\!], p) & = & \langle f \rangle \\ \mathsf{st2g}([\![x, st_1, st_2]\!], p) & = & \textbf{if } x \in \mathsf{P} \textbf{ then } \langle x, \mathsf{st2g}(st_1, p), \mathsf{st2g}(st_2, p)\rangle \\ & & \qquad\qquad\ \textbf{else } \langle p, \mathsf{st2g}(st_1, p), \mathsf{st2g}(st_2, p)\rangle. \end{array}$$

If a strategy st is really the strategy of agent p it should contain nowhere p and should contain a choice c instead. In this case we say that st is *full for* p and we write it $st \overset{p}{\Downarrow}$.

$$[\![f]\!] \overset{p}{\Downarrow}$$
$$[\![x, st_1, st_2]\!] \overset{p}{\Downarrow} \Leftrightarrow^{c} (x \notin \mathsf{Choice} \Rightarrow x \neq p) \wedge st_1 \overset{p}{\Downarrow} \wedge st_2 \overset{p}{\Downarrow}.$$

We can sum strategies to make a strategy profile. But for that we have to assume that all strategies are full and underlie the same game. In other words, $(st_p)_{p\in\mathsf{P}}$ is a family of strategies such that:

- $\forall p \in \mathsf{P}, st_p \overset{p}{\Downarrow}$,
- there exists a game g such that $\forall p \in \mathsf{P}, \mathsf{st2g}(st_p) = g$.

We define $\bigoplus_{p\in\mathsf{P}} st_p$ as follows:

$$\bigoplus_{p\in\mathsf{P}} [\![f]\!] = \langle\!\langle f \rangle\!\rangle$$
$$[\![c, st_{p',1}, st_{p',2}]\!] \oplus \bigoplus_{p\in\mathsf{P}\setminus p'} [\![p', st_{p,1}, st_{p,2}]\!] = \langle\!\langle p', c, \bigoplus_{p\in\mathsf{P}} st_{p,1}, \bigoplus_{p\in\mathsf{P}} st_{p,2}\rangle\!\rangle.$$

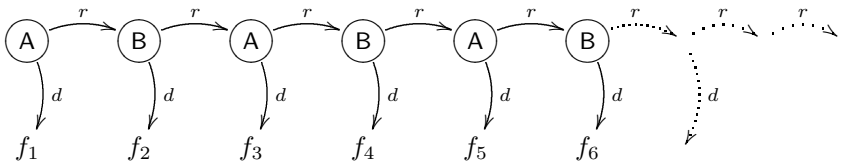
We can show that the game underlying all the strategies is the game underlying the strategy profile which is the sum of the strategies.

Proposition 4. $\mathsf{st2g}(st_{p'}, p') = \mathsf{game}(\bigoplus_{p\in\mathsf{P}} st_p)$.

[3] A strategy is not the same as a strategy profile, which is obtained as the sum of strategies.

3 Comb Games and the 0,1-Game

We will restrict to simple games which have the shape of combs,



At each step the agents have only two choices, namely to stop or to continue. Let us call such a game, a *comb game*.

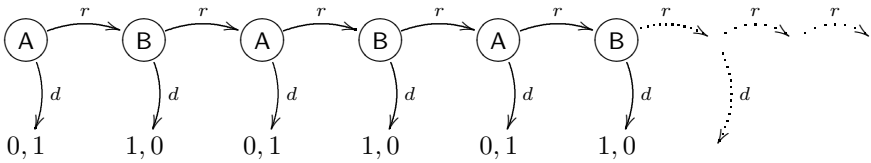
We introduce infinite games by means of equations. Let us see how this applies to define the $0, 1$-game. First consider two payoff functions:

$$f_{0,1} = \mathsf{A} \mapsto 0, \mathsf{B} \mapsto 1$$
$$f_{1,0} = \mathsf{A} \mapsto 1, \mathsf{B} \mapsto 0$$

we define two games

$$G_{0,1} = \langle \mathsf{A}, \langle f_{0,1} \rangle, G_{1,0} \rangle$$
$$G_{1,0} = \langle \mathsf{B}, \langle f_{1,0} \rangle, G_{0,1} \rangle$$

This means that we define an infinite sequential game $G_{0,1}$ in which agent A is the first player and which has two subgames: the trivial game $\langle f_{1,0} \rangle$ and the game $G_{1,0}$ defined in the other equation. The game $G_{0,1}$ can be pictured as follows:



or more simply in Figure 1.a.

From now on, we assume that we consider only strategy profiles whose underlying game is the 0,1-game. They are characterized by the following predicates

$$\mathsf{S}_0(s) \Leftrightarrow_c s = \langle\!\langle \mathsf{A}, c, f_{0,1}, s' \rangle\!\rangle \wedge \mathsf{S}_1(s')$$
$$\mathsf{S}_1(s) \Leftrightarrow_c s = \langle\!\langle \mathsf{B}, c, f_{1,0}, s' \rangle\!\rangle \wedge \mathsf{S}_0(s').$$

Notice that the $0, 1$-game we consider is somewhat a zero-sum game, but we are not interested in this aspect. Moreover, a very specific instance of a $0, 1$ game has been considered (by Ummels [22] for instance), but these authors are not interested in the general structure of the game, but in a specific model on a finite graph, which is not general enough for our taste. Therefore this is not a direct generalization of finite sequential games (replacing induction by coinduction) and this not a framework to study escalation.

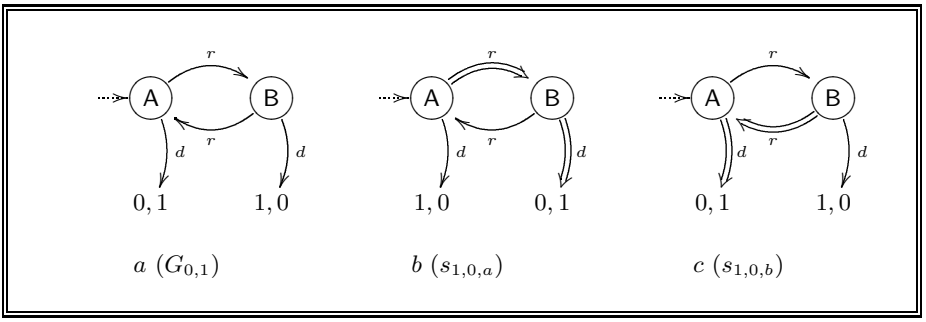


Fig. 1. The 0, 1-game and two equilibria seen compactly

4 Subgame Perfect Equilibria

Among the strategy profiles, one can select specific ones that are called *subgame perfect equilibria*. Subgame perfect equilibria are specific strategy profiles that fulfill a predicate SPE. This predicate relies on another predicate PE which checks a local property.

$$\begin{aligned}\mathsf{PE}(s) \quad \Leftrightarrow \quad s \Downarrow \ &\wedge\ s = \langle\!\langle p, d, s_d, s_r \rangle\!\rangle \Rightarrow \widehat{s_d}(p) \geq \widehat{s_r}(p)\\ &\wedge\ s = \langle\!\langle p, r, s_d, s_r \rangle\!\rangle \Rightarrow \widehat{s_r}(p) \geq \widehat{s_d}(p)\end{aligned}$$

A strategy profile is a subgame perfect equilibrium if the property PE holds always:

$$\mathsf{SPE} = \Box\mathsf{PE}.$$

We may now wonder what the subgame perfect equilibria of the 0,1-game are. We present two of them in Figure 1.b and 1.c. But there are others. To present them, let us define a predicate *"A continues and B eventually stops"*

$$\begin{aligned}\mathsf{AcBes}(s) \Longleftrightarrow s = \langle\!\langle p, c, \langle\!\langle f \rangle\!\rangle, s' \rangle\!\rangle \Rightarrow\ &(p = \mathsf{A} \wedge f = f_{0,1} \wedge c = r \wedge \mathsf{AcBes}(s')) \vee\\ &(p = \mathsf{B} \wedge f = f_{1,0} \wedge (c = d \vee \mathsf{AcBes}(s'))\end{aligned}$$

Proposition 5. $(\mathsf{S}_1(s) \vee \mathsf{S}_0(s)) \Rightarrow \mathsf{AcBes}(s) \Rightarrow \widehat{s} = f_{1,0}$

Proof. If $s = \langle\!\langle p, c, \langle\!\langle f \rangle\!\rangle, s' \rangle\!\rangle$, then $\mathsf{S}_0(s') \vee \mathsf{S}_1(s')$. Therefore if $\mathsf{AcBes}(s')$, by induction, $\widehat{s'} = f_{0,1}$. By case:

- If $p = \mathsf{A} \wedge c = r$, then $\mathsf{AcBes}(s')$ and by definition of $\widehat{s}$, we have $\widehat{s} = \widehat{s'} = f_{0,1}$
- if $p = \mathsf{B} \wedge c = d$, the $\widehat{s} = \widehat{\langle\!\langle f_{0,1} \rangle\!\rangle} = f_{0,1}$.
- if $p = \mathsf{B} \wedge c = r$, , then $\mathsf{AcBes}(s')$ and by definition of $\widehat{s}$, $\widehat{s} = \widehat{s'} = f_{0,1}$.

Like we generalize PE to SPE by applying the modality $\Box$, we generalize AcBes into SAcBes by stating:

$$\mathsf{SAcBes} = \Box\mathsf{AcBes}.$$

There are at least two profiles which satisfies SAcBes namely $s_{1,0,a}$ and $s_{1,0,b}$ which have been studied in [14] and pictured in Figure 1:

$$s_{1,0,a} \overset{c}{\rightleftarrows} \langle\langle \mathsf{A}, r, f_{0,1}, s_{1,0,b} \rangle\rangle \qquad s_{1,0,b} \overset{c}{\rightleftarrows} \langle\langle \mathsf{B}, d, f_{1,0}, s_{1,0,a} \rangle\rangle$$
$$s_{0,1,a} \overset{c}{\rightleftarrows} \langle\langle \mathsf{A}, d, f_{0,1}, s_{0,1,b} \rangle\rangle \qquad s_{0,1,b} \overset{c}{\rightleftarrows} \langle\langle \mathsf{B}, r, f_{1,0}, s_{0,1,a} \rangle\rangle$$

Proposition 6. $\mathsf{SAcBes}(s) \Rightarrow s \Downarrow$.

We may state the following proposition.

Proposition 7. $\forall s, (\mathsf{S}_0(s) \vee \mathsf{S}_1(s)) \Rightarrow (\mathsf{SAcBes}(s) \Rightarrow \mathsf{SPE}(s))$.

Proof. Since SPE is a coinductively defined predicate, the proof is by coinduction.

Given an s, we have to prove $\forall s, \Box\mathsf{AcBes}(s) \wedge (\mathsf{S}_0(s) \vee \mathsf{S}_1(s)) \Rightarrow \Box\mathsf{PE}(s)$.

For that we assume $\Box\mathsf{AcBes}(s) \wedge (\mathsf{S}_0(s) \vee \mathsf{S}_1(s))$ and in addition (coinduction principle) $\Box\mathsf{PE}(s')$ for all strict subprofiles s' of s and we prove $\mathsf{PE}(s)$. In other words, $s \Downarrow \wedge \langle\langle p, d, s_d, s_r \rangle\rangle \Rightarrow \widehat{s_d}(p) \geq \widehat{s_r}(p) \ \wedge \ \langle\langle p, r, s_d, s_r \rangle\rangle \Rightarrow \widehat{s_r}(p) \geq \widehat{s_d}(p)$.

By Proposition 6, we have $s \Downarrow$.

By Proposition 5, we know that for every subprofile s' of a profile s that satisfies $\mathsf{S}_1(s) \vee \mathsf{S}_0(s)$ we have $\widehat{s'} = f_{1,0}$ except when $s' = \langle\langle f_{0,1} \rangle\rangle$. Let us prove $\langle\langle p, d, s_d, s_r \rangle\rangle \Rightarrow \widehat{s_d}(p) \geq \widehat{s_r}(p) \ \wedge \ \langle\langle p, r, s_d, s_r \rangle\rangle \Rightarrow \widehat{s_r}(p) \geq \widehat{s_d}(p)$. Let us proceed by case:

- $s = \langle\langle \mathsf{A}, r, \langle\langle f_{0,1} \rangle\rangle, s' \rangle\rangle$. Then $\mathsf{S}_0(s)$ and $\mathsf{S}_1(s')$. Since $\Box\mathsf{AcBes}(s)$, we have $\mathsf{AcBes}(s')$, therefore $\widehat{s'} = f_{1,0}$ hence $\widehat{s'}(\mathsf{A}) = 1$ and $f_{0,1}(\mathsf{A}) = 0$, henceforth $\widehat{s'}(\mathsf{A}) \geq f_{0,1}(\mathsf{A})$.
- $s = \langle\langle \mathsf{B}, r, \langle\langle f_{1,0} \rangle\rangle, s' \rangle\rangle$. Then $\mathsf{S}_1(s)$ and $\mathsf{S}_0(s')$. Since $\Box\mathsf{AcBes}(s)$, we have $\mathsf{AcBes}(s')$, therefore $\widehat{s'} = f_{1,0}$ hence $\widehat{s'}(\mathsf{B}) = 0$ and $f_{1,0}(\mathsf{B}) = 0$, henceforth $\widehat{s'}(\mathsf{B}) \geq f_{1,0}(\mathsf{B})$.

Symmetrically we can define a predicate BcAes for "B continues and A eventually stops" and a predicate SBcAes which is $\mathsf{SBcAes} = \Box\ \mathsf{BcAes}$ which means that B continues always and A stops infinitely often. With the same argument as for SAcBes one can conclude that

$$\forall s, (\mathsf{S}_0(s) \vee \mathsf{S}_1(s)) \Rightarrow \mathsf{SBcAes}(s) \Rightarrow \mathsf{SPE}(s).$$

We claim that $\mathsf{SAcBes} \vee \mathsf{SBcAes}$ fully characterizes SPE of 0,1-games, in other words.

Conjecture 1. $\forall s, (\mathsf{S}_0(s) \vee \mathsf{S}_1(s)) \Rightarrow (\mathsf{SAcBes}(s) \vee \mathsf{SBcAes} \Leftrightarrow \mathsf{SPE}(s))$.

5 Nash Equilibria

Before talking about escalation, let us see the connection between subgame perfect equilibrium and Nash equilibrium in a sequential game. In [17], the definition of a Nash equilibrium is as follows: *A Nash equilibrium is a "pattern[s] of behavior*

with the property that if every player knows every other player's behavior she has not reason to change her own behavior" in other words, *"a Nash equilibrium [is] a strategy profile from which no player wishes to deviate, given the other player's strategies."* . The concept of deviation of agent p is expressed by a binary relation we call *convertibility*[4] and we write $\vdash^p \dashv$. It is defined inductively as follows:

$$\frac{s \sim_s s'}{s \vdash^p \dashv s'}$$

$$\frac{s_1 \vdash^p \dashv s_1' \qquad s_2 \vdash^p \dashv s_2'}{\langle\!\langle p, c, s_1, s_2 \rangle\!\rangle \vdash^p \dashv \langle\!\langle p, c', s_1', s_2' \rangle\!\rangle}$$

$$\frac{s_1 \vdash^p \dashv s_1' \qquad s_2 \vdash^p \dashv s_2'}{\langle\!\langle p', c, s_1, s_2 \rangle\!\rangle \vdash^p \dashv \langle\!\langle p', c, s_1', s_2' \rangle\!\rangle}$$

We define the predicate Nash as follows:

$$\mathsf{Nash}(s) \Leftrightarrow \forall p, \forall s', s \vdash^p \dashv s' \Rightarrow \widehat{s}(p) \geq \widehat{s'}(p').$$

The concept of Nash equilibrium is more general than that of subgame perfect equilibrium and we have the following result:

Proposition 8. $\mathsf{SPE}(s) \Rightarrow \mathsf{Nash}(s)$.

The result has been proven in COQ and we refer to the script (see[15]):

`http://perso.ens-lyon.fr/pierre.lescanne/COQ/EscRatAI/`
`http://perso.ens-lyon.fr/pierre.lescanne/COQ/EscRatAI/SCRIPTS/`

Notice that we defined the convertibility inductively, but a coinductive definition is possible. But this would give a more restrictive definition of Nash equilibrium.

6 Escalation

Escalation in a game with a set P of agents occurs when there is a tuple of strategies $(st_p)_{p \in \mathsf{P}}$ such that its sum is not convergent, in other words, $\neg\, (\bigoplus_{p \in \mathsf{P}} st_p)\downarrow$. Said differently, it is possible that the agents have all a private strategy which combined with those of the others makes a strategy profile which is not convergent, which means that the strategy profile goes to infinity when following the choices. Notice the two uses of a strategy profile: first, as a subgame perfect equilibrium, second as a combination of the strategies of the agents.

Consider the strategy:

$$st_{\mathsf{A},\infty} = [\![r, [\![f_{0,1}]\!], st'_{\mathsf{A},\infty}]\!]$$
$$st'_{\mathsf{A},\infty} = [\![\mathsf{B}, [\![f_{1,0}]\!], st_{\mathsf{A},\infty}]\!]$$

[4] This should be called perhaps *feasibility* following [18] and [13].

and its twin

$$st_{\mathsf{B},\infty} = [\![\mathsf{A}, [\![f_{0,1}]\!], st'_{\mathsf{B},\infty}]\!]$$
$$st'_{\mathsf{B},\infty} = [\![r, [\![f_{1,0}]\!], st_{\mathsf{B},\infty}]\!].$$

Moreover, consider the strategy profile:

$$s_{\mathsf{A},\infty} = \langle\!\langle \mathsf{A}, r, \langle\!\langle f_{0,1} \rangle\!\rangle, s_{\mathsf{B},\infty} \rangle\!\rangle$$
$$s_{\mathsf{B},\infty} = \langle\!\langle \mathsf{B}, r, \langle\!\langle f_{1,0} \rangle\!\rangle, s_{\mathsf{A},\infty} \rangle\!\rangle.$$

Proposition 9.

- $st_{A,\infty} \overset{\mathsf{A}}{\Updownarrow}$,
- $st_{B,\infty} \overset{\mathsf{B}}{\Updownarrow}$,
- $\mathsf{st2g}(st_{A,\infty}, A) = \mathsf{st2g}(st_{B,\infty}, B) = G_{0,1}$,
- $\mathsf{game}(s_{A,\infty}) = G_{0,1}$,
- $st_{A,\infty} \oplus st_{B,\infty} = s_{A,\infty}$,
- $\neg\, s_{A,\infty} \downarrow$.

Proof. By coinduction.

$st_{\mathsf{A},\infty}$ and $st_{\mathsf{B},\infty}$ are both rational since they are built using choices, namely r, dictated by subgame perfect equilibria[5] which start with r. Another feature of this example is that no agent has a clue for what strategy the other agent is using. Indeed after k steps, A does not know if B has used a strategy derived of equilibria in SAcBes or in SBcAes. In other words, A does know if B will stop eventually or not and vice versa. The agents can draw no conclusion of what they observe. If each agent does not believe in the threat of the other she is naturally led to escalation.

7 Conclusion

In this paper, we have shown how to use coinduction in a field, namely economics, where it has not been used yet, or perhaps in a really hidden form, which has to be unearthed. We foresee a future for this tool and a possible way for a refoundation of economics.

Acknowledgements. The author thanks Samson Abramsky, Franck Delaplace, Stephane Leroux, Matthieu Perrinel, René Vestergaard, Viktor Winschel for their help, encouragements and discussions during this research.

[5] Our choice of rationality is this of a subgame perfect equilibrium, as it generalizes backward induction, which is usually accepted as the criterion of rationality for finite game.

References

1. Abramsky, S., Winschel, V.: Coalgebraic analysis of subgame-perfect equilibria in infinite games without discounting. arXiv (2012)
2. Aumann, R.J.: Backward induction and common knowledge of rationality. Games and Economic Behavior 8, 6–19 (1995)
3. Bouchaud, J.-P.: Economics needs a scientific revolution. Nature 455, 1181 (2008)
4. Bouchaud, J.-P., Potters, M.: Theory of Financial Risk and Derivate Pricing. Cambridge University Press (2003)
5. Capretta, V.: Common knowledge as a coinductive modality. In: Barendsen, E., Geuvers, H., Capretta, V., Niqui, M. (eds.) Reflections on Type Theory, Lambda Calculus, and the Mind, pp. 51–61. ICIS, Faculty of Science, Radbout University Nijmegen (2007), Essays Dedicated to Henk Barendregt on the Occasion of his 60th Birthday
6. Colander, D.: The future of economics: the appropriately educated in pursuit of the knowable. Cambridge Journal of Economics 29, 927–941 (2005)
7. Colman, A.M.: Game theory and its applications in the social and biological sciences, 2nd edn. Routledge, London (1999)
8. Gintis, H.: Game Theory Evolving: A Problem-Centered Introduction to Modeling Strategic Interaction. Princeton University Press (2000)
9. Kuhn, H.W.: Extensive games and the problem of information. In: Contributions to the Theory of Games II (1953), Reprinted in [10]
10. Kuhn, H.W. (ed.): Classics in Game Theory. Princeton Uni. Press (1997)
11. Leininger, W.: Escalation and coooperation in conflict situations. J. of Conflict Resolution 33, 231–254 (1989)
12. Lescanne, P., Perrinel, M.: From coinduction to the rationality of escalation (September 2009), `http://perso.ens-lyon.fr/pierre.lescanne/PUBLICATIONS/escalation_rational.pdf`
13. Lescanne, P.: Feasibility/desirability games for normal form games, choice models and evolutionary games. CoRR, abs/0907.5469 (2009)
14. Lescanne, P.: Rationality and escalation in infinite extensive games. CoRR, abs/1112.1185 (2011)
15. Lescanne, P., Perrinel, M.: "Backward" coinduction, Nash equilibrium and the rationality of escalation. Acta Inf. 49(3), 117–137 (2012)
16. O'Neill, B.: International escalation and the dollar auction. J. of Conflict Resolution 30(33-50) (1986)
17. Osborne, M.J.: An Introduction to Game Theory, Oxford (2004)
18. Rubinstein, A.: Lecture Notes in Microecomic Theory. Princeton University Press (2006)
19. Selten, R.: Spieltheoretische Behandlung eines Oligopolmodells mit Nachfrageträgheit. Zeitschrift für gesamte Staatswissenschaft 121 (1965)
20. Shubik, M.: The dollar auction game: A paradox in noncooperative behavior and escalation. Journal of Conflict Resolution 15(1), 109–111 (1971)
21. Tarski, A.: A lattice-theoretical fixpoint theorem and its applications. Pacific Journal of Mathematics 5(2), 285–309 (1955)
22. Ummels, M.: The complexity of Nash equilibria in infinite multiplayer games. In: Amadio, R.M. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 20–34. Springer, Heidelberg (2008)
23. Vestergaard, R.: A constructive approach to sequential Nash equilibria. Inf. Process. Lett. 97, 46–51 (2006)

24. Weierstrass, K.: Über continuirliche Funktionen eines reellen Arguments, die für keinen Werth des letzteren einen bestimmten Differentialquotienten besitzen. In: Karl Weiertrass Mathematische Werke, Abhandlungen II Gelesen in der Königl. Akademie der Wisseschaften, Juli 18, pp. 71–74 (1872)
25. Winschel, V.: Private communication (2012)

A Finite 0,1 Games and the "Cut and Extrapolate" Method

We spoke about the "cut and extrapolate" method, applied in particular to the dollar auction. Let us see how it would work on the 0,1-game. Finite games, finite strategy profiles and payoff functions of finite strategy profiles are the inductive equivalent of infinite games, infinite strategy profiles and infinite payoff functions which we presented. Notice that payoff functions of finite strategy profiles are always defined. Despite we do not speak of the same types of objects, we use the same notations, but this does not lead to confusion. Consider two finite families of finite games, that could be seen as approximations of the 0,1-game:

$$F_{0,1} = \langle \mathsf{A}, \langle f_{0,1}\rangle, \langle \mathsf{B}, \langle f_{1,0}\rangle, F_{0,1}\rangle\rangle \cup \{\langle f_{0,1}\rangle\} \qquad \begin{array}{l} K_{0,1} = \langle \mathsf{A}, \langle f_{0,1}\rangle, K'_{0,1}\rangle \\ K'_{0,1} = \langle \mathsf{B}, \langle f_{1,0}\rangle, K_{0,1}\rangle \cup \{\langle f_{1,0}\rangle\} \end{array}$$

In $F_{0,1}$ we cut after B and replace the tail by $\langle f_{0,1}\rangle$. In $K_{0,1}$ we cut after A and replace the tail by $\langle f_{1,0}\rangle$. Recall [23] the predicate BI, which is the finite version of PE.

$$\begin{array}{rl} \mathsf{BI}(\langle f\rangle) & \\ \mathsf{BI}(\langle p, c, s_d, s_r\rangle) = & \mathsf{BI}(s_l) \wedge \mathsf{BI}(s_r) \wedge \\ & \langle\!\langle p, d, s_d, s_r\rangle\!\rangle \Rightarrow \widehat{s_d}(p) \geq \widehat{s_r}(p) \ \wedge \\ & \langle\!\langle p, r, s_d, s_r\rangle\!\rangle \Rightarrow \widehat{s_r}(p) \geq \widehat{s_d}(p) \end{array}$$

We consider the two families of strategy profiles:

$$\begin{array}{ll} \mathsf{SF}_{0,1}(s) \Leftrightarrow & (s = \langle\!\langle \mathsf{A}, d \vee r, \langle\!\langle f_{0,1}\rangle\!\rangle, \langle\!\langle \mathsf{B}, r, \langle\!\langle f_{1,0}\rangle\!\rangle, s'\rangle\!\rangle\rangle\!\rangle \wedge \mathsf{SF}_{0,1}(s') \quad \vee \\ & s = \langle\!\langle f_{0,1}\rangle\!\rangle \end{array}$$

$$\begin{array}{ll} \mathsf{SK}_{0,1}(s) \Leftrightarrow & s = \langle\!\langle \mathsf{A}, r, \langle\!\langle f_{0,1}\rangle\!\rangle, s'\rangle\!\rangle \wedge \mathsf{SK}'_{0,1}(s') \\ \mathsf{SK}'_{0,1}(s) \Leftrightarrow & (s = \langle\!\langle \mathsf{B}, d, \langle\!\langle f_{1,0}\rangle\!\rangle, s'\rangle\!\rangle \vee s = \langle\!\langle \mathsf{B}, r, \langle\!\langle f_{1,0}\rangle\!\rangle, s'\rangle\!\rangle) \wedge \mathsf{SK}_{0,1}(s') \quad \vee \\ & s = \langle\!\langle f_{1,0}\rangle\!\rangle \end{array}$$

In $\mathsf{SF}_{0,1}$, B continues and A does whatever she likes and in $\mathsf{SK}_{0,1}$, A continues and B does whatever she likes. The following proposition characterizes the backward induction equilibria for games in $F_{0,1}$ and $K_{0,1}$ respectively and is easily proved by induction:

Proposition 10.

- $\mathsf{game}(s) \in F_{0,1} \wedge \mathsf{SF}_{0,1}(s) \Leftrightarrow \mathsf{BI}(s)$,
- $\mathsf{game}(s) \in K_{0,1} \wedge \mathsf{SK}_{0,1}(s) \Leftrightarrow \mathsf{BI}(s)$.

This shows that cutting at an even or an odd position does not give the same strategy profile by extrapolation. Consequently the "cut and extrapolate" method does not anticipate all the subgame perfect equilibria. Let us add that when cutting we decide which leaf to insert, namely $\langle f_{0,1} \rangle$ or $\langle f_{1,0} \rangle$, but we could do another way around obtaining different results.

0,1 game and limited payroll. To avoid escalation in the dollar auctions, some require a *limited payroll*, i.e., a bound on the amount of money handled by the agents, but this is inconsistent with the fact that the game is infinite. Said otherwise, to avoid escalation, they forbid escalation. One can notice that, in the 0,1-game, a limited payroll would not prevent escalation, since the payoffs are anyway limited by 1.

Coalgebras with Symmetries and Modelling Quantum Systems

Daniel Marsden

University of Oxford

Abstract. This paper describes a generalization of the usual category in which coalgebras are considered, and its application to modelling quantum systems and their physical symmetries. Following the programme of work initiated in [1], [2], we aim to model systems described by the laws of quantum physics using coalgebraic techniques. A broader notion of the morphisms of coalgebras is given, in which diagrams are allowed to commute only up to appropriate natural isomorphism. This relaxed setting is then shown to have analogues of coalgebraic notions such as bisimulations, with properties that parallel the usual coalgebraic ones closely. This new setting is then exploited to give coalgebraic models of quantum systems in which the conceptually important physical symmetries are given as automorphisms of a suitable coalgebra.

Finally, we investigate coalgebraic logic in this setting, showing that there is a natural notion of "symmetry modality" that can be exploited. The notions of Schrödinger and Heisenberg evolution are discussed, and it is argued that Heisenberg evolution is more natural in the coalgebraic setting. It is then shown that these additional modalities can be used to give an adequate and expressive coalgebraic logic for quantum system in which state evolution and measurement outcomes can be described by suitable modal operators. An appropriate model of this logic then gives predictions consistent with the laws of quantum mechanics.

1 Introduction

When describing the behaviour of a quantum system, interactions with the system can be divided into two types:

- Measurements
- Reversible evolution of the system state

Measurement outcomes are probabilistic, and their properties are governed by the important Born rule of quantum mechanics. The reversible evolution of the system state is clearly related to group actions on the underlying mathematical model. This suggests when we model a quantum system coalgebraically we should hope to find the appropriate symmetries as automorphisms of our "quantum" coalgebra.

In categories of coalgebras, morphisms are functional bisimulations, in particular, for a strongly extensional coalgebra in which bisimilarity reduces to

R. Heckel and S. Milius (Eds.): CALCO 2013, LNCS 8089, pp. 205–219, 2013.

equality, there can only be one automorphism. This leads to a tension between the nature of coalgebra homomorphisms and the desire to model symmetries effectively.

To resolve this tension, [2] introduced a novel fibrational structure, in which a representation of the physical symmetries of a quantum system could be found as automorphisms of a particular coalgebra. A similar construction was used in [3] to model parameterization of coalgebras, and logical aspects developed in [4]. In this paper we aim for a technically simpler approach, emphasising the importance of automorphisms of the signature functor as a uniform method of reversibly adapting the dynamics of a given coalgebra. A suitable choice of group of automorphisms can then be chosen to weaken the notion of coalgebra morphism in order provide the flexibility to model symmetries successfully.

Section 2 introduces this weakened category of coalgebras, and shows that notions from conventional coalgebra theory generalize smoothly to this broader category. We then exploit our new setting to provide full and faithful representations of physically relevant groups of symmetries of quantum systems.

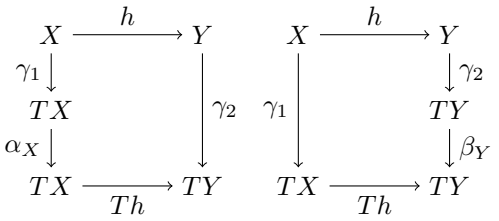
In section 3, we continue to exploit automorphisms of the signature functor to extend the predicate lifting style of coalgebraic logic [5], [6] with suitable symmetry modalities. These symmetry modalities are then used to give a coalgebraic logic for quantum systems capturing both evolution of the system state and probabilistic measurement outcomes in a manner consistent with quantum mechanics.

2 Coalgebras with Symmetries

We now introduce a generalization of the notion of morphism between coalgebras, in which the usual diagram for coalgebra homomorphisms is only required to commute up to appropriate isomorphism. Our general programme is then to extend coalgebraic notions such as bisimulation to this new setting in an appropriate manner. These generalized notions will be given the prefix "pseudo" to distinguish them from the usual terminology. To avoid an excessively technical presentation, we shall often restrict our attention to coalgebras over the category **Set**.

Definition 1. *For category $\mathcal{C}$, endofunctor $T : \mathcal{C} \to \mathcal{C}$ and G a subgroup of of the automorphisms of T (natural isomorphims of type $T \Rightarrow T$), define the category* T-**PseudoCoalg**(G) *as follows:*

- *Objects: T-coalgebras*
- *Morphisms: A morphism $(X, \gamma_1 : X \to TX) \to (Y, \gamma_2 : Y \to TY)$ is a $\mathcal{C}$ morphism $h : X \to Y$ such that equivalently either there exists $\alpha \in G$ making the left hand diagram below commute, or there exists $\beta \in G$ such that the right hand diagram commutes:*



The equivalence of the two conditions can seen by pasting an appropriate naturality square using the inverse natural isomorphisms, on the bottom of each diagram. Identities and composition are as in the base category.

Definition 2. *In* T-**PseudoCoalg**(G)*:*

- *A morphism where the usual coalgebra homomorphism condition holds strictly will be referred to as a* **functional bisimulation**
- *A morphism of the form* $1_X : (X, \gamma) \to (X, \alpha_X \circ \gamma)$ *will be referred to as an* **adaptation**

Lemma 1. *Let* $\mathcal{C}$ *be a category,* $T : \mathcal{C} \to \mathcal{C}$ *an endofunctor, and* G *a subgroup of the automorphisms of* T*:*

1. *Every morphism in* T-**PseudoCoalg**(G) *factors as an adaption followed by a functional bisimulation*
2. *Every morphism in* T-**PseudoCoalg**(G) *factors as a functional bisimulation followed by an adaptation*

Proof. Consider T-**PseudoCoalg**(G) morphism:

$$(X, \gamma_1) \xrightarrow{h} (Y, \gamma_2)$$

Then there exists $\alpha \in G$ such that h is a coalgebra homomorphism $(X, \alpha_X \circ \gamma_1) \to (Y, \gamma_2)$. For part 1 we take the factorization:

$$(X, \gamma_1) \xrightarrow{1} (X, \alpha_X \circ \gamma_1) \xrightarrow{h} (Y, \gamma_2)$$

Also there must exists $\beta \in G$ such that h is a coalgebra homomorphism $(X, \gamma_1) \to (Y, \beta_Y \circ \gamma_2)$. For part 2 we take the factorization:

$$(X, \gamma_1) \xrightarrow{h} (X, \beta_Y \circ \gamma_2) \xrightarrow{1} (Y, \gamma_2)$$

Definition 3 (Pseudo-bisimulation). *For endofunctor* $T : \mathbf{Set} \to \mathbf{Set}$ *and* G *a subgroup of the automorphisms of* T*, a* **pseudo-bisimulation** *between coalgebras* $(X, \gamma_1 : X \to TX)$ *and* $(Y, \gamma_2 : Y \to TY)$ *is a relation* $R \subseteq X \times Y$ *such that the span given by the projection morphisms in* **Set***:*

$$X \xleftarrow{\pi_1} R \xrightarrow{\pi_2} Y$$

lifts to a span between (X, γ_1) *and* (Y, γ_2) *in* T-**PseudoCoalg**(G).

We now outline some simple properties of pseudo-bisimulations, in analogy to [7]. The proofs either follow from the corresponding property of bisimulations, or from a slight modification of the associated proof.

Lemma 2. *For endofunctor* $T : \mathbf{Set} \to \mathbf{Set}$ *and* G *a subgroup of the automorphisms of* T*, if* $h : (X, \gamma_1) \to (Y, \gamma_2)$ *is a morphism in* T-**PseudoCoalg**(G)*, then the graph of* h *is a pseudo-bisimulation.*

Proof. Assume $h : (X, \gamma_1) \to (Y, \gamma_2)$ is a morphism in T-**PseudoCoalg**(G), then there exists $\alpha \in G$ such that we have coalgebra morphism $h : (X, \alpha_X \circ \gamma_1) \to (Y, \gamma_2)$, and so the graph of h gives a bisimulation between these coalgebras, and so a pseudo-bisimulation between (X, γ_1) and (Y, γ_2).

Lemma 3. *For endofunctor* $T : \mathbf{Set} \to \mathbf{Set}$ *and* G *a subgroup of the automorphisms of* T*:*

1. *Every bisimulation is a pseudo-bisimulation*
2. *The inverse of a pseudo-bisimulation is a pseudo-bisimulation*
3. *For morphisms* $f : (X, \gamma_1) \to (Y, \gamma_2)$ *and* $g : (X, \gamma_1) \to (Z, \gamma_3)$*,* $im(\langle f, g \rangle)$ *is a pseudo-bisimulation between* (Y, γ_2) *and* (Z, γ_3)
4. *If* T *preserves weak pullbacks then pseudo-bisimulations compose as relations*
5. *If* T *preserves weak pullbacks then the kernel of a morphism is a pseudo-bisimulation*

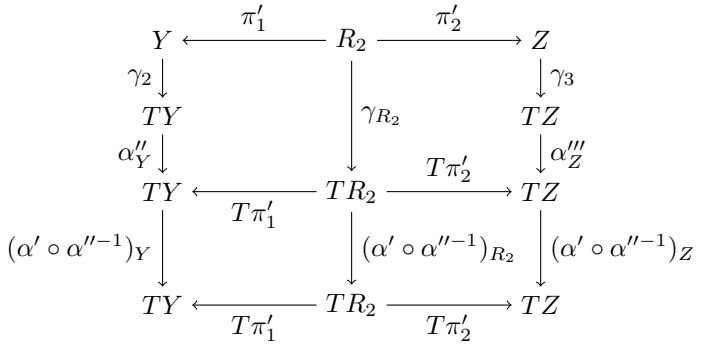
Proof. Parts 1 and 2 are obvious.

For part 3 there must exist $\alpha, \beta \in G$ such that f is a coalgebra morphism $f : (X, \gamma_1) \to (Y, \alpha_Y \circ \gamma_2)$ and g is a coalgebra morphism $g : (X, \gamma_1) \to (Z, \beta_Z \circ \gamma_3)$. Therefore $im(\langle f, g \rangle)$ is a bisimulation between $(Y, \alpha_Y \circ \gamma_2)$ and $(Z, \beta_Z \circ \gamma_3)$, which is then a pseudo-bisimulation between (Y, γ_2) and (Z, γ_3).

For part 4 assume (R_1, γ_{R_1}) is a pseudo-bisimulation between (X, γ_1) and (Y, γ_2), then there exist $\alpha, \alpha' \in G$ such that the following diagram commutes:

$$\begin{array}{ccccc}
X & \xleftarrow{\pi_1} & R_1 & \xrightarrow{\pi_2} & Y \\
{\scriptstyle\gamma_1}\downarrow & & \downarrow{\scriptstyle\gamma_{R_1}} & & \downarrow{\scriptstyle\gamma_2} \\
TX & & & & TY \\
{\scriptstyle\alpha_X}\downarrow & & \downarrow & & \downarrow{\scriptstyle\alpha'_Y} \\
TX & \xleftarrow[T\pi_1]{} & TR_1 & \xrightarrow[T\pi_2]{} & TY
\end{array}$$

Also assume (R_2, γ_{R_2}) is a pseudo-bisimulation between (Y, γ_2) and (Z, γ_3), then there exist $\alpha'', \alpha''' \in G$ such that the following diagram commutes:



Therefore R_1 is a bisimulation between $(X, \alpha_X \circ \gamma_1)$ and $(Y, \alpha'_Y \circ \gamma_2)$, and R_2 is a bisimulation between $(Y, \alpha'_Y \circ \gamma_2)$ and $(Z, (\alpha' \circ \alpha''^{-1} \circ \alpha''')_X \circ \gamma_3)$. We then have a bisimulation between $(X, \alpha_X \circ \gamma_1)$ and $(Z, (\alpha' \circ \alpha''^{-1} \circ \alpha''')_Z \circ \gamma_3)$, and so a pseudo-bisimulation between (X, γ_1) and (Z, γ_3).

Part 5 then follows from parts 2, 4 and lemma 2, as the kernel of a function can be formed as the relational composition of its graph and its converse.

We now note a significant difference between conventional bisimulations and pseudo-bisimulations, essentially caused by the need to choose compatible natural automorphisms of the signature functor, when combining pseudo-bisimulations under union.

Lemma 4. *In general pseudo-bisimulations are not closed under even finite unions.*

Now we will exploit the extra freedom given by the more general class of morphism to capture the physical symmetries of a quantum systems as automorphisms of a suitable coalgebra. Suitable background on quantum computation can be found in [8] or [9].

Definition 4. *Let $\mathcal{H}$ be some Hilbert space. $\mathcal{L}(\mathcal{H})$ will denote the lattice of projection operators on $\mathcal{H}$. Following [2] we define a signature functor for modelling quantum systems as follows:*

$$Q : \mathbf{Set} \to \mathbf{Set} \tag{1}$$

$$Q := (1 + (0, 1] \times 1_{\mathbf{Set}})^{\mathcal{L}(\mathcal{H})} \tag{2}$$

The idea of this signature is that $\mathcal{L}(\mathcal{H})$ describes propositions we can probabilistically test on our system. If a measurement outcome is impossible, this will be modelled by an outcome in the one element set, otherwise we give the probability of the measurement outcome, and the state of the system after the measurement, as in general, in quantum mechanics, this will differ from the initial state.

We note that Q is a polynomial functor, and so will preserve weak pullbacks.

In quantum mechanics, the pure states of a quantum system are described by rays (lines through the origin) in Hilbert space. This leads us to consider a coalgebra over a quotient of the vectors in Hilbert space.

Definition 5 (The Quantum Coalgebra). *For a fixed Hilbert space $\mathcal{H}$, define $\mathcal{H}_0$ as the set of all vectors in $\mathcal{H}$ with the origin removed. We define the equivalence relation on $\mathcal{H}_0$:*

$$|\varphi\rangle \sim |\psi\rangle \text{ iff } \exists c \in \mathbb{C}.|\varphi\rangle = c|\psi\rangle \tag{3}$$

We then define **projective Hilbert space** *$\mathcal{P}(\mathcal{H})$ as the quotient $\mathcal{H}_0/\sim$. Two equivalence classes $[|\varphi\rangle]$ and $[|\psi\rangle]$ are said to be* **orthogonal** *if $|\varphi\rangle$ and $|\psi\rangle$ are orthogonal in Hilbert space. In this case we write:*

$$[|\varphi\rangle] \perp [|\psi\rangle] \tag{4}$$

The **quantum coalgebra** *is then defined as the Q-coalgebra $(\mathcal{P}(\mathcal{H}), \gamma_q)$ describing the measurement behaviour governed by the Born rule of quantum mechanics as follows:*

$$\gamma_q([|\varphi\rangle])(\hat{P}) := \begin{cases} (\frac{\langle\varphi|\hat{P}|\varphi\rangle}{\langle\varphi|\varphi\rangle}, [\hat{P}|\varphi\rangle]) & \text{if } \langle\varphi \mid \hat{P} \mid \varphi\rangle \neq 0 \\ \star & \text{otherwise} \end{cases} \tag{5}$$

This coalgebra is strongly extensional [2].

Having modelled measurement behaviour in the dynamics of a suitable coalgebra, we now move to state evolution of our quantum system. This is described mathematically by unitary and (the possibly slightly less well known) antiunitary transformations of our Hilbert space. In fact, physical predictions are unaffected by multiplication by an arbitrary phase, so we shall be interested in groups with this phase quotiented out.

Definition 6. *For Hilbert space $\mathcal{H}$ we will refer to the group of unitary and antiunitary operators as the* **semiunitary group**. *Define the* **projective semiunitary group** *$\mathcal{P}_S$ as the quotient of the semiunitary group identifying elements that are equal up to a phase. Similarly we define the* **projective unitary group** *$\mathcal{P}_U$ as the same quotient restricted to the unitaries.*

Definition 7. *For object A in some category $\mathcal{C}$, the group of automorphisms of A will be denoted* $\mathrm{auto}(A)$.

Lemma 5. *For Hilbert space $\mathcal{H}$, every semiunitary $\hat{U}$ induces an automorphism a_U of $\mathcal{L}(\mathcal{H})$. If two semiunitaries only differ by a phase, they induce the same automorphism. Also $a_1 = 1$ and $a_{U\circ V} = a_U \circ a_V$.*

Proof. Take the function: $\hat{P} \mapsto \hat{U}\hat{P}\hat{U}^\dagger$ with inverse $\hat{P} \mapsto \hat{U}^\dagger\hat{P}\hat{U}$

Corollary 1. *For Hilbert space $\mathcal{H}$, we have a group homomorphism $\alpha^{(-)} : \mathcal{P}_S \to$* $\mathrm{auto}(Q)$.

Proof. For $[U] \in \mathcal{P}_S$ take α^U as the natural isomorphism given by precomposing a_U, i.e.

$$\alpha^U_X(f)(\hat{P}) = f(\hat{U}\hat{P}\hat{U}^\dagger) \tag{6}$$

Definition 8. *We will denote the image of $\mathcal{P}_S$ under $\alpha^{(-)}$ in* auto(Q) *as $\mathcal{A}_S$, and the image of $\mathcal{P}_U$ under α as $\mathcal{A}_U$.*

We state the following version of Wigner's theorem, in the form given in [1], as it will used required for the subsequent lemma. Further details of modern projective geometry can be found in [10], for example.

Theorem 1 (Wigner's Theorem). *For Hilbert spaces $\mathcal{H}$, $\mathcal{H}'$ with $\mathcal{H}$ of dimensional at least 3, let $f : \mathcal{P}(\mathcal{H}) \to \mathcal{P}(\mathcal{H}')$ be a total map of projective geometries. If f satisfies:*

$$[|\varphi\rangle] \perp [|\psi\rangle] \Rightarrow f([|\varphi\rangle]) \perp f([|\psi\rangle]) \tag{7}$$

then there is a semilinear map $V : \mathcal{H} \to \mathcal{H}'$, with associated homomorphism $\sigma : \mathbb{C} \to \mathbb{C}$, such that:

$$f([|\varphi\rangle]) = [V|\varphi\rangle] \tag{8}$$

and

$$\langle V\varphi \mid V\psi\rangle = \sigma(\langle\varphi \mid \psi\rangle) \tag{9}$$

Further, V is unique up to a phase, and σ is either the identity of complex conjugation, so V is either linear or antilinear.

Lemma 6. *For Hilbert space $\mathcal{H}$, we have group homomorphism $h_{(-)} : \mathcal{P}_S \to$* auto$(\mathcal{P}(\mathcal{H}))$ *given as follows:*

$$h_U([|\psi\rangle]) := [U|\psi\rangle] \tag{10}$$

Furthermore, if $\mathcal{H}$ is of dimension greater than 2, $h_{(-)}$ is injective.

Proof. That this is a well defined group homomorphism is straightforward to check. The last part follows from Wigner's theorem as stated in theorem 1, as the maps h_U induced by unitaries and antiunitaries satisfy the required conditions.

We now show that these induced automorphisms of $\mathcal{P}(\mathcal{H})$ lift to coalgebra homomorphisms.

Proposition 1. *For $[U] \in \mathcal{P}_S$, with definitions as in lemma 6 and corollary 1 α^U is a bijection, and the following diagram commutes.*

$$\begin{array}{ccc}
\mathcal{P}(\mathcal{H}) & \xrightarrow{h_U} & \mathcal{P}(\mathcal{H}) \\
\Big\downarrow{\scriptstyle \gamma_q} & & \Big\downarrow{\scriptstyle \gamma_q} \\
Q(\mathcal{P}(\mathcal{H})) & & Q(\mathcal{P}(\mathcal{H})) \\
 & & \Big\downarrow{\scriptstyle \alpha^U} \\
Q(\mathcal{P}(\mathcal{H})) & \xrightarrow[Q(h_U)]{} & Q(\mathcal{P}(\mathcal{H}))
\end{array}$$

Proof. A straightforward diagram chase expanding the definitions as needed.

Remark 1. The automorphism described in proposition 1 can be seen as encoding the physicsts notion of *covariance*, if both the state and physical quantities are evolved by the same unitary our physical predictions remain as before, as captured by bisimilarity in this setting. We can also view these pseudo-coalgebra automorphisms as capturing Schrödinger evolution of the quantum system, in which states change, but physical quantities (as encoded by the coalgebra) remain fixed.

We are now in a position to give our representation results for the physically significant symmetries of a quantum system. We see that by varying the group of automorphisms of the signature functor, we can "fine tune" the automorphisms of our coalgebraic model. Our weaker notion of morphism makes the coalgebra morphisms in proposition 1 into automorphisms in the pseudo setting.

Proposition 2. *For a Hilbert space $\mathcal{H}$ with dimension greater than 2:*

- *In* T-**PseudoCoalg**($\mathcal{A}_S$) $\mathcal{P}_S$ *is fully and faithfully represented as the automorphisms of* γ_q
- *In* T-**PseudoCoalg**($\mathcal{A}_U$) $\mathcal{P}_U$ *is fully and faithfully represented as the automorphisms of* γ_q

Proof. That we have an injective group homomorphism $\mathcal{P}_S \to \text{auto}(\gamma_q)$ in T-**PseudoCoalg**($\mathcal{A}_S$) follows from lemma 6 and proposition 1. For fullness, as γ_q is strongly extensional there can be at most one morphism $\alpha_{\mathcal{P}(\mathcal{H})} \circ \gamma_q \to \gamma_q$ for each $\alpha \in \mathcal{A}_S$. The case for the restriction to unitaries follows similarly.

3 Coalgebraic Logic

In the literature on coalgebraic logic, for example [5] or [6], it is usual to consider behavioural equivalence [11] rather than bisimilarity. We will now generalize that notion to the pseudo-coalgebraic setting.

Definition 9 (Pseudo-behavioural equivalence). *Let $T : \mathbf{Set} \to \mathbf{Set}$ be an endofunctor, G a subgroup of the automorphisms of T, and $(X_1, \gamma_1), (X_2, \gamma_2)$ T-coalgebras. For $x_1 \in X_1$ and $x_2 \in X_2$, x_1 is said to be* **pseudo-behaviourally equivalent** *to x_2 if there exists coalgebra (Y, γ) and* T-**PseudoCoalg**(G) *morphisms $f : (X_1, \gamma_1) \to (Y, \gamma)$ and $g : (X_2, \gamma_2) \to (Y, \gamma)$ such that $f(x_1) = g(x_2)$.*

Lemma 7. *Let T : **Set** $\to$ **Set** be an endofunctor such that bisimilarity is equivalent to behavioural equivalence, G a subgroup of the automorphisms of T, and $(X_1, \gamma_1), (X_2, \gamma_2)$ T-coalgebras. For $x_1 \in X_1$ and $x_2 \in X_2$, x_1 is pseudo-bisimilar to x_2 if and only if x_1 is pseudo-behaviourally equivalent to x_2.*

Proof. To show pseudo-bisimilarity implies pseudo-behavioural equivalence, if x_1 and x_2 are pseudo-bisimilar then there exist natural transformations α, $\beta \in G$ such that there is a bisimulation between $\alpha_{X_1} \circ \gamma_1$ and $\beta_{X_2} \circ \gamma_2$ relating x_1 and x_2. By assumption there is then a behavioural equivalence between $\alpha_{X_1} \circ \gamma_1$ and

$\beta_{X_2} \circ \gamma_2$ relating x_1 and x_2 and so there is a pseudo-behavioural equivalence relating x_1 and x_2.

The other direction follows similarly.

Corollary 2. *If T preserves weak pullbacks then pseudo-bisimilarity is equivalent to pseudo-behavioural equivalence.*

Throughout this section we shall be considering modalities provided by predicate liftings, and to avoid excessive notation we shall restrict our attention to unary modalities, although it is straightforward to extend the ideas discussed to support general arities of operators. As we only consider unary modalities, a **modal signature** Λ is simply a set of (unary) modality symbols. We begin by investigating some basic properties of formulae whose validity is preserved under adaptions.

Definition 10. *For modal signature Λ, and regular cardinal κ, the language $\mathcal{L}^\kappa(\Lambda)$ is defined by the grammar:*

$$\mathcal{L}^\kappa = \bigwedge \Phi \text{ for } \Phi \subseteq \mathcal{L}^\kappa \text{ and } \mathrm{card}(\Phi) < \kappa \mid \neg\mathcal{L}^\kappa \mid \Box_\lambda \mathcal{L}^\kappa \text{ for } \Box_\lambda \in \Lambda \tag{11}$$

For the language $\mathcal{L}^\kappa(\Lambda)$ we will use the usual semantics for coalgebraic modal logic based on predicate liftings.

Definition 11 (Symmetric formulae). *Let $T : \mathbf{Set} \to \mathbf{Set}$ be an endofunctor and G a subgroup of the automorphisms of T. For a given modal signature Λ and choice of predicate liftings, a formula $\varphi \in \mathcal{L}^\kappa(\Lambda)$ will be said to be* **symmetric** *if for each T-coalgebra (X, γ):*

$$\forall x \in X, \forall \alpha, \beta \in G. (X, \alpha_X \circ \gamma), x \models \varphi \text{ iff } (X, \beta_X \circ \gamma), x \models \varphi \tag{12}$$

Lemma 8. *Let $T : \mathbf{Set} \to \mathbf{Set}$ be an endofunctor and G a subgroup of the automorphisms of T. For a given modal signature and choice of predicate liftings:*

1. *$\top$ and $\bot$ are symmetric*
2. *If φ is symmetric then $\neg\varphi$ is symmetric*
3. *If φ and ψ are symmetric then $\varphi \wedge \psi$ is symmetric*
4. *If φ and ψ are symmetric then $\varphi \vee \psi$ is symmetric*

Proposition 3. *Let $T : \mathbf{Set} \to \mathbf{Set}$ be an endofunctor, and G a subgroup of the automorphisms of T. For a given modal signature and choice of predicate liftings, pseudo-behaviourally equivalent states satisfy the same symmetric formulae.*

Proof. Let $(X_1, \gamma_1), (X_2, \gamma_2)$ be two coalgebras, and assume $x_1 \in X_1$ and $x_2 \in X_2$ are pseudo-behaviourally equivalent, and that φ is a symmetric formula. Then there exist $\alpha, \beta \in G$ such that there is a behavioural equivalence between $\alpha_X \circ \gamma_1$ and $\beta_Y \circ \gamma_2$ relating x_1 and x_2. Now if $\gamma_1, x_1 \models \varphi$ then by symmetry of φ, $\alpha_X \circ \gamma_1 \models \varphi$. Then as behaviourally equivalent element satisfy the same formulae, $\beta_Y \circ \gamma_2, y \models \varphi$, and applying symmetry of φ again $\gamma_2, y \models \varphi$.

We now adopt a slightly different perspective, and aim to exploit the natural isomorphisms that have been central to our work so far, in order to provide additional modalities in our logic. We define an extension of coalgebraic logic based on predicate liftings with additional symmetry operators.

Definition 12. *For group G, modal signature Λ and regular cardinal κ, the language $\mathcal{L}^{\kappa}_{\mathrm{sym}}(\Lambda)$ of* **coalgebraic logic with symmetry modalities** *is defined by the grammar:*

$$\mathcal{L}^{\kappa}_{\mathrm{sym}} = \bigwedge \Phi \text{ for } \Phi \subseteq \mathcal{L}^{\kappa}_{\mathrm{sym}} \text{ and } \mathrm{card}(\Phi) < \kappa \mid \neg\mathcal{L}^{\kappa}_{\mathrm{sym}} \tag{13}$$

$$\mid \Box_\lambda \mathcal{L}^{\kappa}_{\mathrm{sym}} \text{ for } \Box_\lambda \in \Lambda \tag{14}$$

$$\mid [g]\mathcal{L}^{\kappa}_{\mathrm{sym}} \text{ for } g \in G \tag{15}$$

Definition 13 (Semantics of Coalgebraic Logic with Symmetry Modalities). *Let $T : \mathbf{Set} \to \mathbf{Set}$ be an endofunctor, G a group together with a group homomorphism $[\![-]\!] : G \to \mathrm{auto}(T)$ and (X, γ) a T-coalgebra. We consider a modal signature $\Lambda = \{\Box_\lambda\}$ with semantics given by predicate liftings $[\![\Box_\lambda]\!] : 2 \Rightarrow 2 \circ T$. We define the semantics for $\mathcal{L}^{\kappa}_{\mathrm{sym}}(\Lambda)$ as follows:*

$$[\![\bigwedge \Phi]\!]_{X,\gamma} := \bigcap_{\varphi \in \Phi} [\![\varphi]\!]_{X,\gamma} \tag{16}$$

$$[\![\neg\varphi]\!]_{X,\gamma} := X \setminus [\![\varphi]\!]_{X,\gamma} \tag{17}$$

$$[\![\Box_\lambda\varphi]\!]_{X,\gamma} := \gamma^{-1} \circ [\![\Box_\lambda]\!]_X([\![\varphi]\!]_{X,\gamma}) \tag{18}$$

$$[\![[g]\varphi]\!]_{X,\gamma} := [\![\varphi]\!]_{X,[\![g]\!]_X \circ \gamma} \tag{19}$$

The semantics given above are identical to the standard formulation, with the exception of the new symmetry modalities.

Remark 2. In proposition 1 we described automorphisms corresponding to semi-unitary evolution of the quantum coalgebra and how these corresponded to Schrödinger evolution in physics. When extending coalgebraic logic with symmetry modalities, we wish to be able to interpret these modalities on every coalgebra of the given signature, whereas the existence of automorphisms such as those in proposition 1 is specific to the coalgebra in question. This leads us to the semantics of symmetry modalities given above, where we instead adapt the dynamics of the coalgebra using the appropriate natural isomorphism, leaving the underlying states fixed. In the quantum case, we can see this as a Heisenberg type evolution, in which states remain fixed and physical quantities evolve with time. So it seems Heisenberg evolution is more natural than Schrödinger evolution in the coalgebraic setting.

Lemma 9. *For definitions as in definition 13, symmetry operators have the following properties for arbitrary formula φ:*

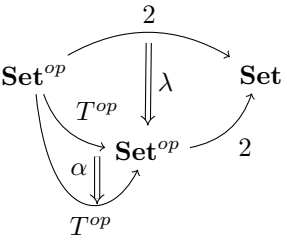
- *For 1 the unit element of G, $[1]\varphi$ is logically equivalent to φ*
- *For all $g, g' \in G$, $[g][g']\varphi$ is logically equivalent to $[g' \circ g]\varphi$*

Proof. Immediate from the definitions.

Lemma 10. *For definitions as in definition 13 and symmetric formula φ, for all $g, g' \in G$, $[g]\varphi$ is logically equivalent to $[g']\varphi$.*

Proof. Immediate from the definitions.

Definition 14. *Let $2 : \mathbf{Set}^{op} \to \mathbf{Set}$ be the contravariant powerset functor. For predicate lifting $\lambda : 2 \Rightarrow 2 \circ T^{op}$, and α a natural isomorphism of T we define predicate lifting λ^α as the pasting:*



We now introduce some additional modalities that will be exploited to relate our modal logic with symmetry modalities to standard coalgebraic modal logic with predicate liftings.

Definition 15. *For a modal signature Λ, and group G together with group homomorphism $[\![-]\!] : G \to \mathrm{auto}(T)$, we define the* **extended modal signature***:*

$$\Lambda^G := \{\Box_\lambda^g \mid g \in G, \Box_\lambda \in \Lambda\} \tag{20}$$

Given a corresponding predicate liftings $[\![\Box_\lambda]\!] : 2 \Rightarrow 2 \circ T^{op}$, the **extended liftings** *for the modalities above are given by $[\![\Box_\lambda^g]\!] = [\![\Box_\lambda]\!]^{[\![g]\!]}$ using the pastings in definition 14.*

We now show that given symmetry modalities, the modalities given by the extended predicate liftings of definition 15 can be defined syntactically.

Lemma 11. *Let $T : \mathbf{Set} \to \mathbf{Set}$ be an endofunctor, G a subgroup of natural isomorphisms of T and modal signature Λ. For modality $\Box_\lambda \in \Lambda$ with predicate lifting $[\![\Box_\lambda]\!]$ and modality $\Box_\lambda^g \in \Lambda^G$ with semantics given as in definition 15, for arbitrary formula $\varphi \in \mathcal{L}^\kappa_{\mathrm{sym}}(\Lambda^G)$:*

$$[\![\Box_\lambda^g \varphi]\!]_{X,\gamma} = [\![[g]\Box_\lambda [g^{-1}]\varphi]\!]_{X,\gamma} \tag{21}$$

Proof.

$$\begin{aligned}
[\![[g]\Box_\lambda [g^{-1}]\varphi]\!]_{X,\gamma} &= [\![\Box_\lambda [g^{-1}]\varphi]\!]_{X,[\![g]\!]_X \circ \gamma} && (22)\\
&= ([\![g]\!]_X \circ \gamma)^{-1} \circ [\![\Box_\lambda]\!]([\![[g^{-1}]\varphi]\!]_{X,[\![g]\!]_X \circ \gamma}) && (23)\\
&= ([\![g]\!]_X \circ \gamma)^{-1} \circ [\![\Box_\lambda]\!]([\![\varphi]\!]_{X,\gamma}) && (24)\\
&= \gamma^{-1} \circ [\![\Box_\lambda]\!]^{[\![g]\!]}([\![\varphi]\!]_{X,\gamma}) && (25)\\
&= \gamma^{-1} \circ [\![\Box_\lambda^g]\!]([\![\varphi]\!]_{X,\gamma}) && (26)\\
&= [\![\Box_\lambda^g \varphi]\!]_{X,\gamma} && (27)
\end{aligned}$$

Now we proceed in the opposite direction, showing how the symmetry operators can be replaced by appropriate use of the modalities in the extended signature.

Definition 16. *For modal signature Λ, group G and $g \in G$ we define syntactic transformation τ_g from $\mathcal{L}^{\kappa}_{\mathrm{sym}}(\Lambda)$ to $\mathcal{L}^{\kappa}(\Lambda^G)$ inductively:*

$$\tau_g(\bigwedge \Phi) := \bigwedge \{\tau_g(\varphi) \mid \varphi \in \Phi\} \tag{28}$$

$$\tau_g(\neg\varphi) := \neg\tau_g(\varphi) \tag{29}$$

$$\tau_g(\Box_\lambda \varphi) := \Box^g_\lambda \tau_g(\varphi) \tag{30}$$

$$\tau_g([g']\varphi) := \tau_{g' \circ g}(\varphi) \tag{31}$$

We now relate the semantics of $\mathcal{L}^{\kappa}_{\mathrm{sym}}(\Lambda)$ given in definition 13 to the usual coalgebraic logic semantics of $\mathcal{L}^{\kappa}(\Lambda^G)$, via the translation τ_g.

Theorem 2. *Let $T : \mathbf{Set} \to \mathbf{Set}$ be an endofunctor, G a group together with a group homomorphism $[\![-]\!] : G \to \mathrm{auto}(T)$ and (X, γ) a T-coalgebra. Let $\Lambda = \{\Box_\lambda\}$ be a modal signature with the semantics for each modality denoted $[\![\Box_\lambda]\!] : 2 \Rightarrow 2 \circ T$. We have for formula $\varphi \in \mathcal{L}^{\kappa}_{\mathrm{sym}}$ and $g \in G$:*

$$[\![\varphi]\!]_{X, g_X \circ \gamma} = [\![\tau_g(\varphi)]\!]_{X,\gamma} \tag{32}$$

Proof. The proof is by induction on the structure of formulae, the cases for the Boolean connectives are straightforward. We will explicitly show the cases for the two types of modalities.

Firstly, for symmetry modalities we have:

$$[\![\tau_g([g']\varphi)]\!]_{X,\gamma} = [\![\tau_{g' \circ g}(\varphi)]\!]_{X,\gamma} \tag{33}$$

$$= [\![\varphi]\!]_{X, [\![g]\!]'_X \circ [\![g]\!]_X \circ \gamma} \tag{34}$$

$$= [\![[g']\varphi]\!]_{X, g_X \circ \gamma} \tag{35}$$

where the second equality follows by the induction hypothesis.

For the usual modalities we have the following sequence of equalities:

$$[\![\Box_\lambda \varphi]\!]_{X, g_X \circ \gamma} = \gamma^{-1} \circ [\![g]\!]^{-1}_X \circ [\![\Box_\lambda]\!]_X([\![\varphi]\!]_{[\![g]\!]_X \circ \gamma}) \tag{36}$$

$$= \gamma^{-1} \circ [\![\Box^g_\lambda]\!]_X([\![\varphi]\!]_{X, [\![g]\!]_X \circ \gamma}) \tag{37}$$

$$= \gamma^{-1} \circ [\![\Box^g_\lambda]\!]_X([\![\tau_g(\varphi)]\!]_{X,\gamma}) \tag{38}$$

$$= [\![\Box^g_\lambda \tau_g(\varphi)]\!]_{X,\gamma} \tag{39}$$

$$= [\![\tau_g(\Box_\lambda \varphi)]\!]_{X,\gamma} \tag{40}$$

where the third equality follows from the induction hypothesis.

Corollary 3. *In coalgebraic logic with symmetry operators, behavioural equivalence implies logical equivalence.*

Proof. Immediate as every formulae in coalgebraic logic with symmetry operators is equivalent to a formula in standard coalgebraic logic without symmetry operators, and this respects behavioural equivalence. (See for example [5]).

Lemma 12. *For Hilbert space $\mathcal{H}$, the functor Q given in definition 4 is κ-accessible where κ is the successor cardinal of* $\mathrm{card}(\mathcal{L}(\mathcal{H}))$.

Proposition 4. *For the functor Q given in definition 4, we have:*

- *For each $\hat{P} \in \mathcal{L}(\mathcal{H})$ a predicate lifting $[\![\Box_{\hat{P}}]\!]$ defined by:*

$$[\![\Box_{\hat{P}}]\!]_X(U) := \{f \in Q(X) \mid \exists r \in \mathbb{R}, u \in U. f(\hat{P}) = (r, u)\} \tag{41}$$

- *For each $\hat{P} \in \mathcal{L}(\mathcal{H})$ and $r \in \mathbb{R}$ a predicate lifting $[\![\Box_{\hat{P},r}]\!]$ defined by:*

$$[\![\Box_{\hat{P},r}]\!]_X(U) := \{f \in Q(X) \mid \exists u \in U. f(\hat{P}) = (r, u)\} \tag{42}$$

Furthermore these predicate liftings:

- *Are monotone.*
- *Are continuous.*
- *Give a separating set of liftings for Q.*

Proof. The naturality of the predicate liftings, monotonicity and continuity are straightforward to check. That these lifings are separating follows from a case analysis on the structure of $Q(X)$ for an arbitrary set X.

Finally we can give an adequate and expressive (infinitary) logic for coalgebras of the quantum signature functor with modalities capturing both probabilistic measurement behaviour and state evolution. In the case of the quantum coalgebra γ_q the semantics will agree with the predictions of quantum mechanics by construction.

Theorem 3. *Let $\mathcal{H}$ be some Hilbert space, Q be the functor in definition 4, and G the subgroup of automorphisms of Q induced by semiunitary transformations. For modal signature Λ corresponding to the predicate liftings in proposition 4, the language $\mathcal{L}^{\kappa}_{\mathrm{sym}}(\Lambda)$ gives an adequate, expressive monotone and normal logic, where κ is the successor cardinal of* $\mathrm{card}(\mathcal{L}(\mathcal{H}))$.

Proof. Adequacy comes from theorem 2. For expressiveness, by lemma 12 and proposition 4 we can apply theorem 14 of [6]. Also by proposition 4, theorems 27 and 30 of [6] give monotonicity and normality.

4 Conclusion and Future Work

We have shown that introducing a more relaxed notion of homomorphism of coalgebras provides the freedom to model symmetries in a flexible manner. This was demonstrated in the case of quantum systems where either the projective unitary group or the projective semiunitary group could be represented fully and faithfully as automorphisms of a suitable coalgebra.

If we pursue this line of thinking further, we could consider "lax" morphisms of coalgebras in which we permit a suitable choice of general natural transformations rather than just isomorphisms, potentially between varying signature

functors. This would move us closer to the fibred setting of [2]. In the case of quantum systems, such a setting would allow convenient handling of composite systems, the importance of which is heavily emphasised in categorical quantum mechanics [12].

The work described here extended the predicate lifting approach to coalgebraic logic with symmetry modalities, enabling the construction of a logic for quantum systems describing both state evolution and probabilistic measurement behaviour. This work should be expanded to cover other formulations such as Moss style coalgebraic logic [13] and duality based approaches [14], [15], [16]. As the introduction of symmetry modalities was a relatively lightweight extension to the semantics of the logic, exploiting further extensions such as fixed point operators [17] should be explored, particularly in the quantum setting. Once we further loosen the notion of coalgebra morphism to a "lax" setting, more possibilities arise in the logical aspects of coalgebra. A suitable form of "fibred coalgebraic logic" is currently being investigated by the author. Finally, these generalized coalgebraic logics should be exploited to analyze standard quantum protocols such as teleporation [18], logic gate teleporation [19], entanglement swapping [20] and key exchange [21], strengthening the connections between quantum computation and the coalgebraic techniques of computer science.

Acknowledgements. Feedback from and discussions with Samson Abramsky and Andreas Döring are gratefully acknowledged. I would also like to thank the anonymous referees for their details comments and suggestions.

References

1. Abramsky, S.: Big toy models: Representing physical systems as Chu spaces. CoRR abs/0910.2393 (2009)
2. Abramsky, S.: Coalgebras, Chu spaces, and representations of physical systems. In: LICS, pp. 411–420. IEEE Computer Society (2010)
3. Kurz, A., Pattinson, D.: Notes on coalgebras, cofibrations and concurrency. Electr. Notes Theor. Comput. Sci. 33, 196–229 (2000)
4. Kurz, A., Pattinson, D.: Coalgebras and modal logic for parameterized endofunctors. Technical Report SEN-R0040, CWI (2000)
5. Pattinson, D.: Coalgebraic modal logic, soundness, completeness and decidability of local consequence. Theoretical Computer Science 309, 177–193 (2003)
6. Schröder, L.: Expressivity of coalgebraic modal logic: The limits and beyond. Theor. Comput. Sci. 390(2-3), 230–247 (2008)
7. Rutten, J.J.M.M.: Universal coalgebra: a theory of systems. Theor. Comput. Sci. 249(1), 3–80 (2000)
8. Mermin, N.: Quantum Computer Science: An Introduction. Cambridge University Press (2007)
9. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information. 10th anniversary edn. Cambridge University Press (2010)
10. Stubbe, I., van Steirteghem, B.: Propositional systems, Hilbert lattices and generalized Hilbert spaces. In: Ensegger, K., Gabbay, D.M., Lehmann, D. (eds.) Handbook of Quantum Logic and Quantum Structures: Quantum Structures, pp. 477–523. Elsevier (2007)

11. Kurz, A.: Logics for Coalgebras and Applications to Computer Science. PhD thesis, Ludwig-Maximilians-Universität München (2000)
12. Abramsky, S., Coecke, B.: Categorical quantum mechanics. In: Engesser, K., Gabbay, D.M., Lehmann, D. (eds.) Handbook of Quantum Logic and Quantum Structures. Elsevier (2008)
13. Moss, L.: Coalgebraic logic. Ann. Pure Appl. Logic 96 (1999)
14. Abramsky, S.: A cook's tour of the finitary non-well-founded sets. CoRR abs/1111.7148 (2011)
15. Kupke, C., Kurz, A., Venema, Y.: Stone coalgebras. Electr. Notes Theor. Comput. Sci. 82(1), 170–190 (2003)
16. Klin, B.: Coalgebraic modal logic beyond sets. Electr. Notes Theor. Comput. Sci. 173, 177–201 (2007)
17. Cîrstea, C., Kupke, C., Pattinson, D.: EXPTIME tableaux for the coalgebraic mu-calculus. Logical Methods in Computer Science 7(3) (2011)
18. Bennett, C.H., Brassard, G., Crépeau, C., Jozsa, R., Peres, A., Wooters, W.K.: Teleporting an unknown state via dual classical and Einstein-Podolsky-Rosen channels. Physical Review Letters 70, 1895–1899 (1993)
19. Gottesman, D., Chang, I.: Quantum teleporation is a universal computational primitive. Nature 402, 390–393 (1999)
20. Źukowski, M., Zeilinger, A., Horne, M., Ekert, A.: "Event ready detectors" Bell experiments via entanglement swapping. Physical Review Letters 71, 4287–4290 (1993)
21. Ekert, A.K.: Quantum cryptography based on Bell's theorem. Physical Review Letters 67, 661–663 (1991)

From Operational Chu Duality to Coalgebraic Quantum Symmetry

Yoshihiro Maruyama*

Quantum Group, Dept. of Computer Science, University of Oxford

Abstract. We pursue the principles of duality and symmetry building upon Pratt's idea of the Stone Gamut and Abramsky's representations of quantum systems. In the first part of the paper, we first observe that the Chu space representation of quantum systems leads us to an operational form of state-observable duality, and then show via the Chu space formalism enriched with a generic concept of closure conditions that such operational dualities (which we call "T_1-type" as opposed to "sober-type") actually exist in fairly diverse contexts (topology, measurable spaces, and domain theory, to name but a few). The universal form of T_1-type dualities between point-set and point-free spaces is described in terms of Chu spaces and closure conditions. From the duality-theoretical perspective, in the second part, we improve upon Abramsky's "fibred" coalgebraic representation of quantum symmetries, thereby obtaining a finer, "purely" coalgebraic representation: our representing category is properly smaller than Abramsky's, but still large enough to accommodate the quantum symmetry groupoid. Among several features, our representation reduces Abramsky's two-step construction of his representing category into a simpler one-step one, thus rendering the Grothendieck construction redundant. Our purely coalgebraic representation stems from replacing the category of sets in Abramsky's representation with the category of closure spaces in the light of the state-observable duality telling us that closure is a right perspective on quantum state spaces.

1 Introduction

It is not uncommon these days to hear of applications of (the methods of) theoretical computer science to foundations of quantum physics; broadly speaking, theoretical computer science seems to be taking steps towards a new kind of "pluralistic unified science" (not monistic one in logical positivism) via the language and methodology of category theory. Among them, Abramsky [1,2] represents quantum systems as Chu spaces and as coalgebras, giving striking characterisations of quantum symmetries based upon the classic Wigner Theorem. Revisiting his work, in the present paper, we develop a Chu-space-based theory of dualities encompassing a form of state-observable duality in quantum physics, and thereafter improve upon his coalgebraic characterisation of quantum symmetries from our duality-theoretical perspective, in order to exhibit the meaning of duality.

* I am grateful to Samson Abramsky and Bob Coecke for discussions and hospitalities. This work was supported by the Nakajima Foundation.

R. Heckel and S. Milius (Eds.): CALCO 2013, LNCS 8089, pp. 220–235, 2013.

In Pratt's Stone Gamut paper [16], he analyses Stone-type dualities in the language of Chu spaces, saying boldly, but with good reasons, "the notoriously difficult notion of Stone duality reduces simply to matrix transposition." The concept of Chu spaces has played significant roles in fairly broad contexts, including concurrency and semantics of linear logic; similar concepts have been used in even more diverse disciplines, like Barwise-Seligman's classifications, Sambin's formal topology and basic pairs, Scott's information systems, and state-property systems in quantum foundations. This work is inspired by Pratt's perspective on Chu spaces, extending the realm of duality theory built upon the language of Chu spaces by enriching it with a generic concept of closure conditions.

In general, we have two types of dualities, namely sober-type and T_1-type ones, between set-theoretical concepts of space and their point-free, algebraic abstractions, which shall be called point-set spaces and point-free spaces respectively. The difference between the two types of dualities in fact lies in the difference between maximal and primal spectra. Our duality theory in this paper focuses upon T_1-type dualities between point-set and point-free spaces. The logical concept of closure conditions is contrived to the end of treating different sorts of point-set and point-free spaces in a unified manner, allowing us to discuss at once topological spaces, measurable spaces, closure spaces, convexity spaces, and so fourth. In a nutshell, the concept of closure conditions prescribes the notion of space. Whilst a typical example of sober-type duality is the well-known duality between sober spaces and spatial frames, an example of T_1-type duality is a duality between T_1 closure spaces and atomistic meet-complete lattices, including as particular instances state-observable dualities between quantum state spaces (with double negation closures) and projection operator lattices in the style of operational quantum mechanics (see Coecke and Moore [5] or Moore [14]).

Our theory of T_1-type dualities enables us to derive a number of concrete T_1-type dualities in various contexts, which include T_1-type dualities between Scott's continuous lattices and convexity spaces, between σ-complete boolean algebras and measurable spaces, and between topological spaces and frames, to name but a few. Let us illustrate by a topological example a striking difference between sober-type and T_1-type dualities. The T_1-type duality in topology is a duality between T_1 spaces and coatomistic frames in which continuous maps correspond not to frame homomorphisms but to maximal homomorphisms, which are frame homomorphisms $f : L \to L'$ such that, given a maximal join-complete ideal $M \subset L'$, $f^{-1}(M)$ is again a maximal join-complete ideal. Although the duality for T_1 spaces is not mentioned in standard references such as Johnstone [9], nevertheless, we consider it important for the reason that some spaces of interest are not sober but T_1: e.g., affine varieties in k^n with k an ACF (i.e., algebraically closed field) are non-sober T_1 spaces (if they are not singletons). Note that Bonsangue et al. [4] shows a duality for T_1 spaces via what they call observation frames, which are frames with additional structures, yet the T_1-type duality above only relies upon plain frame structures.

Whilst sober-type dualities are based upon prime spectrum "Spec", T_1-type dualities are based upon maximal spectrum "Spm". Different choices of spectrum lead to different Chu representations of algebras A concerned: maximal spectrum gives $(A, \mathrm{Spec}(A), e)$ and prime spectrum gives $(A, \mathrm{Spm}(A), e)$ where e is two-valued and defined in both cases by: $e(a, M) = 1$ iff $a \in M$. Accordingly, the corresponding classes of Chu morphisms are distinctively different: e.g., in locale theory, the Spec-based representation characterises frame homomorphisms as Chu morphisms (as shown in Pratt [16]), and the Spm-based representation characterises maximal homomorphisms as Chu morphisms (as shown in this paper for general point-free spaces encompassing frames as just a particular instance). In this way, the Chu space formalism yields a natural account of why different concepts of homomorphisms appear in sober-type and T_1-type dualities.

As in the case above, Chu morphisms can capture different sorts of homomorphisms by choosing different Chu representations. This is true even in quantum contexts, and in particular we can represent quantum symmetries as Chu morphisms by a suitable Chu representation. Coalgebras are Chu spaces with dynamics, and we have a coalgebraic representation of quantum symmetries as well. To be precise, in moving from Chu space to coalgebras, Abramsky [2] relies upon a fibred category $\int \mathbf{F}$ obtained by gluing categories $\mathbf{Coalg}(F^Q)$'s for every $Q \in \mathbf{Set}$ where F^Q is an endofunctor on $\mathbf{Set}$. He uses the Grothendieck construction to "accommodate contravariance" within a coalgebraic framework, fully embedding the groupoid of symmetries into the fibred category $\int \mathbf{F}$.

Looking at the $\int \mathbf{F}$ representation from a duality-theoretical perspective, we consider it odd that there is no structural relationship taken into account between quantum state spaces and projection operator lattices: both are seen as mere sets. For the very reason, Q (which is a projection lattice in a quantum context) first have to be fixed in the endofunctor F^Q on $\mathbf{Set}$ (objects of which are state spaces in a quantum context), and thereafter $\mathbf{Coalg}(F^Q)$'s are glued together to accommodate contravariance regarding $Q \in \mathbf{Set}$. This two-step construction is reduced in the present paper into a simpler, one-step one as follows.

First of all, there is actually a dual, structural relationship between quantum state spaces and projection lattices with the latter re-emerging as the fixpoints (or algebras) of double negation closures (or monads) on the former. This means that Q above can be derived, rather than independently assumed, from a closure structure, if one works on the base category of closure spaces, rather than mere sets. The closure-based reformulation of the $\int \mathbf{F}$ representation leads us to a "Born" endofunctor $\mathbf{B}$ on closure spaces, and to its coalgebra category $\mathbf{Coalg}(\mathbf{B})$, which turns out to be strictly smaller than fairly huge $\int \mathbf{F}$, but still large enough to represent the quantum symmetry groupoid, thus yielding a purely coalgebraic representation and enabling to accommodate contravariance within the single colagebra category $\mathbf{Coalg}(\mathbf{B})$ rather than the fibred $\int \mathbf{F}$ glueing different $\mathbf{Coalg}(F^Q)$'s for all sets Q; notice that contravariance is incorporated into the dualisation process of taking the fixpoints (or algebras) of closures.

2 Duality and Chu Space Representation

We first review basic concepts and notations on Chu spaces and closure spaces.

Chu Spaces. Let us fix a set Ω. A Chu space over Ω is a triple (S, A, e) where S and A are sets, and e is a map from $S \times A$ to Ω. Ω is called the value set, and e the evaluation map. A Chu morphism from (S, A, e) to (S', A', e') is a tuple (f^*, f_*) of two maps $f^* : S \to S'$ and $f_* : A' \to A$ such that $e(x, f_*(a')) = e'(f^*(x), a')$. The category of Chu spaces and Chu morphisms is self-dual, and forms a $*$-autonomous category, giving a fully complete model of linear logic.

For a Chu space $(S, A, e : S \times A \to \Omega)$ and $a \in A$, $e(\text{-}, a) : S \to \Omega$ is called a column of (S, A, e). We denote the set of all columns of (S, A, e) by $\mathrm{Col}(S, A, e)$. On the other hand, $e(x, \text{-}) : A \to \Omega$ is called a row of (S, A, e). We denote the set of all rows of (S, A, e) by $\mathrm{Row}(S, A, e)$. If Ω is ordered, then we equip $\mathrm{Col}(S, A, e)$ and $\mathrm{Row}(S, A, e)$ with the pointwise orderings: e.g., in the case of $\mathrm{Col}(S, A, e)$, this means that, for $a, b \in A$, $e(\text{-}, a) \leq e(\text{-}, b)$ iff $e(x, a) \leq e(x, b)$ for any $x \in S$.

A Chu space (S, A, e) is called extensional iff all the columns are mutually different, i.e., if $e(x, a) = e(x, b)$ for any $x \in S$ then $a = b$. On the other hand, a Chu space (S, A, e) is called separated iff all the rows are mutually different, i.e., if $e(x, a) = e(y, a)$ for any $a \in A$ then $x = y$.

Closure Spaces. Closure spaces may be seen as either a set with a closure operator or a set with a family of subsets that is closed under arbitrary intersections. We denote by $\mathcal{C}(S)$ the set of closed subsets of a closure space S, and by cl(-) the closure operator of S. In this paper we always assume $\emptyset \in \mathcal{C}(S)$ or equivalently $\mathrm{cl}(\emptyset) = \emptyset$. Note then that there is a unique closure structure on a singleton. A map $f : S \to S'$ is called closure-preserving iff $f^{-1}(C) \in \mathcal{C}(S)$ for any $C \in \mathcal{C}(S')$ iff $f(\mathrm{cl}(A)) \subset \mathrm{cl}(f(A))$. We denote by **Clos** the category of closure spaces and closure-preserving maps, which has products and coproducts. A closure space is called T_1 iff any singleton is closed.

2.1 Chu Representation of Quantum Systems

Abramsky [2] represents a quantum system as a Chu space defined via the Born rule, which provides the predictive content of quantum mechanics. Given a Hilbert space H, he constructs the following Chu space over the unit interval $[0, 1]$: $(\mathrm{P}(H), \mathrm{L}(H), e_H : \mathrm{P}(H) \times \mathrm{L}(H) \to [0, 1])$ where $\mathrm{P}(H)$ denotes the set of quantum states as rays (i.e., one-dimesional subspaces) in H, $\mathrm{L}(H)$ denotes the the set of projection operators (or projectors) on H, and finally the evaluation map e_H is defined as follows (let $[\varphi] = \{\alpha\varphi \mid \alpha \in \mathbb{C}\}$): $e_H([\varphi], P) = \frac{\langle\varphi|P\varphi\rangle}{\langle\varphi|\varphi\rangle}$.

We consider that Chu spaces have built-in dualities, or they are dualities without structures: whilst S and A have no structure, e still specifies duality. The category of Chu spaces has duals in terms of monoidal categories; this is internal duality. Can we externalise internal duality in Chu spaces by restoring structures on S and A through e? It is an inverse problem as it were. In the

quantum context, it amounts to explicating the structures of $\mathrm{P}(H)$ and $\mathrm{L}(H)$ that give (external) duality.

The first observation is the bijective correspondences: $\mathrm{P}(H) \simeq \{e(\varphi, \text{-}) \mid \varphi \in \mathrm{P}(H)\} \simeq \{c \in \mathrm{Col}(\mathrm{P}(H), \mathrm{L}(H), e_H) \mid \text{the precisely one 1 appears in } c\}$. So, the states are the atoms of $\mathrm{L}(H)$: in this way we can recover $\mathrm{P}(H)$ from $\mathrm{L}(H)$. This means $\mathrm{L}(H)$ should be equipped with the lattice structure as in Birkhoff-von-Neumann's quantum logic. Although we have $\mathrm{L}(H) \simeq \{e(\text{-}, P) \mid P \in \mathrm{L}(H)\}$, it is not clear at this stage what intrinsic structure of $\mathrm{P}(H)$ enables to recover $\mathrm{L}(H)$ from $\mathrm{P}(H)$. Let us see that a double negation operator on $\mathrm{P}(H)$ does the job.

Define $(\text{-})^{\perp} : \mathcal{P}(\mathrm{P}(H)) \to \mathcal{P}(\mathrm{P}(H))$ as follows: for $X \subset \mathrm{P}(H)$, let $X^{\perp} = \{[\varphi] \in \mathrm{P}(H) \mid \forall [\psi] \in \mathrm{P}(H)\ \langle\varphi|\psi\rangle = 0\}$. It is straightforward to see that $(\text{-})^{\perp\perp}$ is a closure operator on $\mathrm{P}(H)$. Categorically, $(\text{-})^{\perp\perp}$ is a sort of double negation monad. Taking the closed sets or algebras of $(\text{-})^{\perp\perp}$ enables us to recover $\mathrm{L}(H)$:

Proposition 1. *The lattice of closed subsets of* $\mathrm{P}(H)$*, i.e.,* $\{X \subset \mathrm{P}(H) \mid X^{\perp\perp} = X\}$*, is isomorphic to* $\mathrm{L}(H)$*. Schematically,* $\mathcal{C}(\mathrm{P}(H)) \simeq \mathrm{L}(H)$*.*

We thus have a duality between $\mathrm{P}(H)$ qua closure space and $\mathrm{L}(H)$ qua lattice. We can reconstruct $\mathrm{P}(H)$ from $\mathrm{L}(H)$ by taking the atoms on the one hand, and $\mathrm{L}(H)$ from $\mathrm{P}(H)$ by taking the closed sets (or algebras) of $(\text{-})^{\perp\perp}$ on the other. This dualising construction generally works for T_1 closure spaces and atomistic meet-complete lattices, in particular including $\mathrm{P}(H)$ and $\mathrm{L}(H)$ respectively; orthocomplements can be added to this duality.

Categorically, we have a dual equivalence between the category of T_1 closure spaces with closure-preserving maps and the category of atomistic meet-complete lattices with maximal homomorphisms (defined below). This duality is basically known at the object level in operational quantum mechanics (see Moore [14] or Coecke and Moore [5]); nevertheless, our dualisation of arrows, i.e., the concept of maximality, may be new. In this section we aim at developing a theory of such T_1-type dualities in full generality, thereby deriving T_1-type dualities in various concrete contexts as immediate corollaries (which include the state-projector duality). We embark upon this enterprise in the next subsection.

2.2 Chu Theory of T_1-Type Dualities via Closure Conditions

In the following part of this section, we consider two-valued Chu spaces $(S, A, e : S \times A \to \mathbf{2})$ only, where $\mathbf{2}$ denotes $\{0, 1\}$ (with ordering $0 < 1$). This is because in the duality between states $\mathrm{P}(H)$ and property observables $\mathrm{L}(H)$ we do not need other intermediate values in $[0, 1]$; when considering duality, it suffices to take into account whether a value equals 1 or not. On the other hand, intermediate values in $[0, 1]$ play an essential role in characterising quantum symmetries coalgebraically; we need at least three values (i.e., 1, 0, and "neither 0 nor 1"). In a nutshell, duality is possibilistic, whilst symmetry is probabilistic.

In this subsection, we think of (Chu representations of) "point-set" spaces $(S, \mathcal{F})$ where $\mathcal{F} \subset \mathcal{P}(S)$, and of their "point-free" abstractions L which do not have an underlying set S whilst keeping algebraic structures corresponding to

closure properties of $\mathcal{F}$. Especially, we discuss **Top**, **Set**, **Clos**, **Conv**, and **Meas** where **Conv** denotes the category of convexity spaces, which are sets S with $\mathcal{C} \subset \mathcal{P}(S)$ closed under arbitrary intersections and directed unions (quite some convex geometry can be developed based upon such abstract structures; see, e.g., van de Vel [18]); **Meas** denotes the category of measurable spaces, which are sets with $\mathcal{B} \subset \mathcal{P}(S)$ closed under complements and countable intersections.

Morphisms in all of these categories of point-set spaces are defined in the same way as continuous maps, closure-preserving maps, and measurable maps (a.k.a. Borel functions): i.e., they are $f : (S, \mathcal{F}) \to (S', \mathcal{F}')$ such that $f^{-1}(X) \in \mathcal{F}$ for any $X \in \mathcal{F}'$. Note that **Set** may be seen as the category of $(S, \mathcal{F})$ such that $\mathcal{F}$ is maximally closed, i.e., $\mathcal{F} = \mathcal{P}(S)$, with "continuous" maps as morphisms; in such a situation, any map satisfies the condition that $f^{-1}(X) \in \mathcal{F}$ for $X \in \mathcal{F}'$.

Their point-free counterparts are respectively: **Frm** (frames), **CABA** (complete atomic boolean algebras), **MCLat** (meet-complete lattices), **ContLat** (Scott's continuous lattices), and σ**BA** (σ-complete boolean algebras). Continuous lattices may be defined as meet-complete lattices with directed joins distributing over arbitrary meets (this is equivalent to the standard definition via way-below relations; see [6, Theorem I-2.7]); in the light of this, we see continuous lattices as point-free convexity spaces; later, duality justifies this view.

We emphasise that closure conditions on each type of point-set structures correspond to (possibly infinitary) algebraic operations on each type of point-free structures. An insight from our theory is that such a relationship between point-set and point-free spaces always leads us to duality; indeed, we shall show T_1-type dualities between **Top** and **Frm**; **Set** and **CABA**; **Clos** and **MCLat**; **Conv** and **ContLat**; **Meas** and σ**BA**; and even more (e.g., dcpos).

In order to treat different sorts of point-set spaces in a unified manner, we introduce a concept of closure conditions. A closure condition on $\mathcal{F} \subset \mathcal{P}(S)$ is a formula of the following form:

$$\forall \mathcal{X} \subset \mathcal{F}\ (\varphi(\mathcal{X}) \Rightarrow \mathrm{BC}(\mathcal{X}) \in \mathcal{F})$$

where $\mathrm{BC}(\mathcal{X})$ is a (possibly infinitary) boolean combination of elements of $\mathcal{X}$ and $\varphi(\mathcal{X})$ is a closed formula in the language of propositional connectives, quantifiers, equality, a binary, inclusion predicate $\subset$, and nullary, cardinality predicates[1], $\mathrm{card}_{\leq\kappa}(\mathcal{X})$ and $\mathrm{card}_{\geq\kappa}(\mathcal{X})$, for each countable cardinal κ; you may include arbitrary cardinals, though the language becomes uncountable. The domain of the intended interpretation of this language is $\mathcal{X}$, and predicates are to be interpreted in the obvious way: $X \subset Y$ with $X, Y \in \mathcal{X}$ is interpreted as saying that X is a subset of Y, $\mathrm{card}_{\leq\kappa}(\mathcal{X})$ as saying that the cardinality of $\mathcal{X}$ is less than or equal to κ, and so fourth. Note that predicates $\mathrm{card}_{=\kappa}(\mathcal{X})$, $\mathrm{card}_{<\kappa}(\mathcal{X})$, and $\mathrm{card}_{>\kappa}(\mathcal{X})$ are definable in the above language.

[1] First-order logic allows us to express "there are n many elements" for each positive integer n, but cannot express certain cardinality statements (e.g., "there are at most countably many elements"; we need this when defining measurable spaces). For the very reason, we expand the language with the afore-mentioned cardinality predicates.

In this setting, for example, measurable spaces are $(S, \mathcal{F})$ such that $\mathcal{F} \subset \mathcal{P}(S)$ satisfies the following closure conditions: $\forall \mathcal{X} \subset \mathcal{F}\ (\mathrm{card}_{\leq \omega}(\mathcal{X}) \Rightarrow \bigcap \mathcal{X} \in \mathcal{F})$ and $\forall \mathcal{X} \subset \mathcal{F}\ (\mathrm{card}(\mathcal{X}) = 1 \Rightarrow \mathcal{X}^c \in \mathcal{F})$ where $\mathcal{X}^c$ denotes the complement of the unique element of $\mathcal{X}$. and notice that by letting $\mathcal{X} = \emptyset$ we have $\bigcap \emptyset = S \in \mathcal{F}$. Likewise, convexity spaces are $(S, \mathcal{F})$ with $\mathcal{F}$ satisfying the following: $\forall \mathcal{X} \subset \mathcal{F}\ (\top \Rightarrow \bigcap \mathcal{X} \in \mathcal{F})$ and $\forall \mathcal{X} \subset \mathcal{F}\ (\text{“}\mathcal{X} \text{ is directed w.r.t. } \subset \text{”} \Rightarrow \bigcup \mathcal{X} \in \mathcal{F})$ where $\top$ is any tautology and “$\mathcal{X}$ is directed w.r.t. $\subset$” is expressed as “$\forall X\, \forall Y\, \exists Z\ (X \subset Z \wedge Y \subset Z)$”. It is straightforward to find closure conditions for other sorts of point-set spaces. We denote by $\mathfrak{X}_{\mathrm{top}}$ the closure conditions for **Top**, by $\mathfrak{X}_{\mathrm{meas}}$ those for **Meas**, by $\mathfrak{X}_{\mathrm{clos}}$ those for **Clos**, and by $\mathfrak{X}_{\mathrm{conv}}$ those for **Conv**.

Let us denote by $\mathfrak{X}$ a class of closure conditions, and $(S, \mathcal{F})$ with $\mathcal{F} \subset \mathcal{P}(S)$ satisfying $\mathfrak{X}$ is called a point-set $\mathfrak{X}$-space. We always assume that $\mathfrak{X}$ contains: $\forall \mathcal{X} \subset \mathcal{F}\ (\mathrm{card}_{=0}(\mathcal{X}) \Rightarrow \bigcup \mathcal{X} \in \mathcal{F})$. This ensures that $\emptyset$ is in $\mathcal{F}$. We denote by $\mathbf{PtSp}_{\mathfrak{X}}$ the category of point-set $\mathfrak{X}$-spaces with $\mathfrak{X}$-preserving maps (i.e., maps $f : (S, \mathcal{F}) \to (S', \mathcal{F}')$ such that $f^{-1}(X) \in \mathcal{F}$ for any $X \in \mathcal{F}'$). If this setting looks too abstract, $\mathbf{PtSp}_{\mathfrak{X}}$ in the following discussion may be thought of as any of our primary examples: **Top**, **Clos**, **Conv**, and **Meas**.

It plays a crucial role in our duality theory that φ in a closure condition can be interpreted in a point-free setting: in other words, it only talks about the mutual relationships between elements of $\mathcal{X}$, and does not mention elements of elements of $\mathcal{X}$ or any point of an observable region $X \in \mathcal{X}$ (which may be an open set, convex set, measurable set, or the like), thus allowing us to interpret it in any abstract poset $(L, \leq)$ by interpreting the subset symbol $\subset$ as a partial order $\leq$, and lead to the concept of point-free $\mathfrak{X}$-spaces as opposed to point-set ones. We call this interpretation of φ in a poset $(L, \leq)$ the point-free interpretation of φ. Note that the above language for φ is actually nothing but the language of the first-order theory of posets enriched with the cardinality predicates.

A point-set $\mathfrak{X}$-space $(S, \mathcal{F})$ can be regarded as a Chu space $(S, \mathcal{F}, e_{(S,\mathcal{F})} : S \times \mathcal{F} \to \mathbf{2})$ where e is defined by: $e_{(S,\mathcal{F})}(x, X) = 1$ iff $x \in X$.

A special focus of the paper is on T_1 point-set spaces: a point-set $\mathfrak{X}$-space $(S, \mathcal{F})$ is T_1 iff any singleton is in $\mathcal{F}$. When applying this definition to topology, we see a topological space as a set with a family of closed sets rather than open sets. The T_1 property of a Chu space is defined as follows.

Definition 2. *A Chu space (S, A, e) is called T_1 iff for any $x \in S$, there is $a \in A$ such that $e(x, a) = 1$ and $e(y, a) = 0$ for any $y \neq x$.*

Intuitively, a above may be thought of as a region in which there is only one point, namely x, or a property that x does satisfy and any other $y \in S$ does not.

Lemma 3. *A point-set $\mathfrak{X}$-space $(S, \mathcal{F})$ is T_1 iff the corresponding Chu space $(S, \mathcal{F}, e_{(S,\mathcal{F})})$ defined above is a T_1 Chu space.*

Lemma 4. *For point-set $\mathfrak{X}$-spaces $(S, \mathcal{F})$ and $(S', \mathcal{F}')$, a tuple of maps $(f, g) : (S, \mathcal{F}, e_{(S,\mathcal{F})}) \to (S', \mathcal{F}', e_{(S',\mathcal{F}')})$ is a Chu morphism iff $g = f^{-1} : \mathcal{F}' \to \mathcal{F}$ iff $f : (S, \mathcal{F}) \to (S', \mathcal{F}')$ is $\mathfrak{X}$-preserving.*

Lemma 5. *If a Chu space (S, A, e) is* T_1 *and extensional, then for any $x \in S$ there is a unique $a \in A$ such that $e(x, a) = 1$ and $e(y, a) = 0$ for any $y \neq x$.*

Each column $e(\text{-}, a)$ of a Chu space (S, A, e) can be regarded as a subset of S, i.e., as $\{x \in S \mid e(x, a) = 1\}$. We say that $\mathrm{Col}(S, A, e)$ satisfies closure conditions iff the corresponding family of subsets of S satisfies them. The same property can be defined for $\mathrm{Row}(S, A, e)$ as well. The following proposition shows that a broad variety of point-set spaces can be represented as Chu spaces.

Proposition 6. *The category* $\mathbf{PtSp}_{\mathfrak{X}}$ *is equivalent to the category of extensional Chu spaces (S, A, e) such that* $\mathrm{Col}(S, A, e)$ *satisfies the closure conditions $\mathfrak{X}$, denoted by* $\mathbf{ExtChu}_{\mathfrak{X}}$*. In particular, this can be instantiated for $\mathfrak{X}_{\mathrm{top}}$, $\mathfrak{X}_{\mathrm{meas}}$, $\mathfrak{X}_{\mathrm{clos}}$, and $\mathfrak{X}_{\mathrm{conv}}$.*

In the following, we focus on a more specific class of closure conditions. A closure condition $\forall \mathcal{X} \subset \mathcal{F}\ (\varphi(\mathcal{X}) \to \mathrm{BC}(\mathcal{X}) \in \mathcal{F})$ is called pure iff $\mathrm{BC}(\mathcal{X})$ contains precisely one of unions, intersections, and complements. A pure closure condition is monolithic, and does not blend different operations; this is true in any major example mentioned above.

In order to define point-free $\mathfrak{X}$-spaces, we let $\mathfrak{X}$ be a class of pure closure conditions satisfying the following: if a closure condition in $\mathfrak{X}$ contains complementation in its boolean combination part, then the following two closure conditions are in $\mathfrak{X}$: $\forall \mathcal{X} \subset \mathcal{F}\ (\mathrm{card}_{<\omega}(\mathcal{X}) \to \bigcap \mathcal{X} \in \mathcal{F})$ and $\forall \mathcal{X} \subset \mathcal{F}\ (\mathrm{card}_{<\omega}(\mathcal{X}) \to \bigcup \mathcal{X} \in \mathcal{F})$. These additional conditions ensure that once we have complementation on the point-set side we can define boolean negation on the point-free side. Note that, although complementation on sets is, and should be, interpreted as boolean negation on posets of subsets, nevertheless, we are not excluding intuitionistic negation (or interiors of complements of opens), which does not arise from complements in closure conditions (i.e., complements without interiors are boolean), but from unions and finite intersections in them, by which we can define intuitionistic implication, and so intuitionistic negation.

We then define a point-free $\mathfrak{X}$-space as a bounded poset $(L, \leq, 0, 1)$ satisfying the following. If a closure condition in $\mathfrak{X}$ have unions (intersections, complements) in its $\mathrm{BC}(\mathcal{X})$ under the condition φ, then we require L to have joins (meets, boolean negation) under the point-free interpretation of φ (i.e., the subset symbol $\subset$ is interpreted as $\leq$). If one closure condition in $\mathfrak{X}$ contains unions and another contains intersections under the conditions $\varphi(\mathcal{X})$ and $\psi(\mathcal{X})$ respectively, then we require L to satisfy the following (possibly infinitary) distributive law: for any doubly indexed family $\{x_{i,j} \mid i \in I,\ j \in J_i\} \subset L$ with $F := \prod_{i \in I} J_i$, if $\{x_{i,j} \mid j \in J_i\}$ denoted by L_1 and $\{\bigwedge_{i \in I} x_{i,f(i)} \mid f \in F\}$ denoted by L_2 satisfy $\varphi(L_1)$ and $\varphi(L_2)$ respectively, and if $\{x_{i,f(i)} \mid i \in I\}$ denoted by L_3 and $\{\bigvee_{j \in J_i} x_{i,j} \mid i \in I\}$ denoted by L_4 satisfy $\psi(L_3)$ and $\psi(L_4)$ respectively, then $\bigwedge_{i \in I} \bigvee_{j \in J_i} x_{i,j} = \bigvee_{f \in F} \bigwedge_{i \in I} x_{i,f(i)}$. Note that this reduces to the ordinary infinite distributive law in the case of frames, and to distributivity between meets and directed joins in the case of continuous lattices.

There is a subtlety in defining maps f preserving possibly partial operations: e.g., even if $\bigwedge X$ is defined, $\bigwedge f(X)$ is not necessarily defined. In the case of

directed joins of continuous lattices, however, this causes no problem, since directedness is preserved under monotone maps, i.e., if X is directed then $\bigwedge f(X)$ is directed as well. This is also true in the case of σ-complete boolean algebras, since $\mathrm{card}_{\leq\omega}(\text{-})$ is always preserved. With these in mind, we assume: φ in each closure condition in $\mathfrak{X}$ is preserved under monotone maps, i.e., for a monotone map $f : L \to L'$ between point-free $\mathfrak{X}$-spaces L and L', if $\varphi(X)$ holds for $X \subset L$ then $\varphi(f(X))$ holds as well. Homomorphisms of point-free $\mathfrak{X}$-spaces are defined as monotone maps preserving (in general partial) operations induced from the closure conditions in $\mathfrak{X}$. The category of point-free $\mathfrak{X}$-spaces and homomorphisms is denoted by $\mathbf{PfSp}_{\mathfrak{X}}$.

For a point-free $\mathfrak{X}$-space L, we denote the set of atoms in L by $\mathrm{Spm}(L)$, which is called the maximal spectrum of L for the following reason. In the cases of **Frm**, **ContLat**, and **MCLat**, $\mathrm{Spm}(L)$ is actually isomorphic to the maximal filters or ideals with suitable completeness conditions; furthermore, the maximal spectrum of the coordinate ring of an affine variety V in k^n with k an ACF is homeomorphic to $\mathrm{Spm}(L)$ by taking L to be the closed set lattice of V. To exemplify the meaning of "completeness conditions", let us consider **MCLat**. A meet-complete filter is defined as a filter that is closed under arbitrary meets. Since the meet-complete filters of $L \in \mathbf{MCLat}$ bijectively correspond to the principal filters of L, we have an isomorphism between $\mathrm{Spm}(L)$ and the maximal meet-complete filters of L, which holds even in the presence of natural closure structures on them. Alternatively, we may also define $\mathrm{Spm}(L) = \{\uparrow a \mid a \text{ is an atom}\}$ where $\uparrow a = \{x \in L \mid a \leq x\}$. This definition is sometimes more useful than the former.

The continuous maps between T_1 spaces (e.g., affine varieties in $\mathbb{C}^n$) do not correspond to the frame homomorphisms between their open set frames, but to a more restricted class of frame homomorphisms; this exhibits a sharp difference from the case of sober spaces. A maximal homomorphism of point-free $\mathfrak{X}$-spaces is a homomorphism $f : L \to L'$ of them satisfying the maximality condition: for any $b \in \mathrm{Spm}(L')$ there is $a \in \mathrm{Spm}(L)$ such that $\uparrow a = f^{-1}(\uparrow b)$, where note that such an $a \in \mathrm{Spm}(L)$ is necessarily unique. If $\mathrm{Spm}(L)$ is defined as $\{\uparrow a \mid a \text{ is an atom}\}$, then we may state maximality in a more familiar manner: $f^{-1}(M) \in \mathrm{Spm}(L)$ for any $M \in \mathrm{Spm}(L')$. The category of atomistic point-free $\mathfrak{X}$-spaces and maximal homomorphisms is denoted by $\mathbf{AtmsPfSp}_{\mathfrak{X}}$ where recall that a poset with the least element is called atomistic iff any element can be described as the join of a set of atoms. Note that atomic posets and atomistic posets are different in general.

The atomisticity of a Chu space is defined in the following way.

Definition 7. *A Chu space (A, S, e) is called atomistic iff there are $A' \subset A$ and a bijection $\eta : S \to A'$ such that*

1. *any two elements of $\mathrm{Row}(A', S, e')$ are incomparable (with respect to its pointwise ordering) where e' is defined by $e'(a, x) = e(a, x)$;*
2. *for any $x \in S$ and $a \in A$, $e(a, x) = 1$ iff $e(\eta(x), \text{-}) \leq e(a, \text{-})$.*

The intended meaning of A' above is $\mathrm{Spm}(A)$, or the set of atoms of A. In the context of quantum mechanics, item 1 above means that any two quantum

states, when seen as one-dimensional subspaces or projectors onto them, are incomparable, and item 2 means that there is a canonical correspondence between the quantum state space $\mathrm{P}(H)$ and the projection lattice $\mathrm{L}(H)$, by mapping the quantum states to the atoms of the lattice.

Proposition 8. *A Chu space* (S, A, e) *is* T_1 *and extensional iff its dual* $(A, S, \hat{e})$ *is atomistic and separated where we define* $\hat{e}(a, x) = e(x, a)$.

It does not necessarily hold that (S, A, e) is T_1 iff $(A, S, \hat{e})$ is atomistic. As a corollary of the above proposition, we obtain:

Corollary 9. *If a Chu space* (A, S, e) *is atomistic and separated, and* $\mathrm{Row}(A, S, e)$ *has a least element, then* $\mathrm{Row}(A, S, e)$ *is an atomistic poset with its atoms given by* $\{e(\eta(x), \text{-}) \mid x \in S\}$.

Given a point-free $\mathfrak{X}$-space L, we can construct a Chu space $(L, \mathrm{Spm}(L), e_L)$ where e_L is defined by: $e_L(b, a) = 1$ iff $a \leq b$. If we define $\mathrm{Spm}(L) = \{\uparrow a \mid a \text{ is an atom}\}$, the corresponding e_L is specified by: $e_L(a, M) = 1$ iff $a \in M$.

Lemma 10. *A point-free* $\mathfrak{X}$*-space* L *is atomistic iff* $(L, \mathrm{Spm}(L), e_L)$ *is an atomistic Chu space.*

If we define $\mathrm{Spm}(L) = \{\uparrow a \mid a \text{ is an atom}\}$, we can take $\tilde{f}$ in the following lemma to be f^{-1}; in this case, the alternative definition of $\mathrm{Spm}(L)$ seems more transparent than the definition of it as the set of atoms themselves.

Lemma 11. *Let* L *and* L' *be atomistic point-free* $\mathfrak{X}$*-spaces. A pair of maps,* $(f, g) : (L, \mathrm{Spm}(L), e_L) \to (L', \mathrm{Spm}(L'), e_{L'})$*, is a Chu morphism iff* f *is a maximal homomorphism and* $g = \tilde{f}$ *where* $\tilde{f} : \mathrm{Spm}(L') \to \mathrm{Spm}(L)$ *is such that, for any* $b \in \mathrm{Spm}(L')$*,* $f^{-1}(\uparrow b) = \uparrow \tilde{f}(b)$ *(note* $\tilde{f}$ *is well defined because* f *is maximal).*

Proposition 12. *The category* $\mathbf{AtmsPfSp}_{\mathfrak{X}}$ *is equivalent to the category of atomistic separated Chu spaces* (A, S, e) *such that* $\mathrm{Row}(A, S, e)$ *satisfies the closure conditions* $\mathfrak{X}$*, denoted by* $\mathbf{AtmsSepChu}_{\mathfrak{X}}$.

We finally lead to the main duality theorem, exposing and unifying T_1-type dualities in diverse contexts, including sets, topology, measurable spaces, closure spaces, domain theory, and convex geometry.

Theorem 13. $\mathbf{T_1ExtChu}_{\mathfrak{X}}$ *is dually equivalent to* $\mathbf{AtmsSepChu}_{\mathfrak{X}}$*; therefore,* $\mathbf{T_1PsSp}_{\mathfrak{X}}$ *is dually equivalent to* $\mathbf{AtmsPfSp}_{\mathfrak{X}}$*. In particular, this universal duality can be instantiated for* $\mathfrak{X}_{\mathrm{top}}$*,* $\mathfrak{X}_{\mathrm{meas}}$*,* $\mathfrak{X}_{\mathrm{clos}}$*, and* $\mathfrak{X}_{\mathrm{conv}}$.

Although many sorts of point-free spaces are complete, nevertheless, the case of $\mathfrak{X}_{\mathrm{meas}}$ is different, and only requires σ-completeness. In this case, the universal duality above yields a duality between atomistic σ-complete boolean algebras and T_1 measurable spaces. As noted above, **Set** may be seen as the category of $(S, \mathcal{P}(S))$'s with measurable maps (note any map is measurable on $(S, \mathcal{P}(S))$), so that the duality for measurable spaces turns out to restrict to the classic Stone

duality between **Set** and **CABA** (note "atomic" and "atomistic" are equivalent in boolean algebras). It is thus a vast globalisation of the classic Stone duality.

Furthermore, we can apply the theorem above to dcpos (with 0), which is not complete in general, by considering closure under directed unions, which yields point-set spaces $(S, \mathcal{F})$ with $\mathcal{F}$ closed under directed unions; dcpos are their duals. Likewise, preframes fall into the picture as well. We are able to derive even more dualities in the same, simple way; although some general theories of dualities require much labour in deriving concrete dualities (this is a typical complaint on abstract duality theory from the practicing duality theorist), the universal duality above immediately gives us concrete dualities of T_1-type.

The duality obtained in the case of $\mathfrak{X}_{\mathrm{top}}$ is not subsumed by the orthodox duality between sober spaces and spatial frames, since "sober" does not imply "T_1"; there are important examples of non-sober T_1 spaces, including affine varieties in k^n with the Zariski topologies where k is an ACF. As discussed in the Introduction, furthermore, the morphism part of the T_1-type duality is distinctively different from that of the sober-type one.

In the case of $\mathfrak{X}_{\mathrm{conv}}$, we obtain a duality between atomistic continuous lattices and T_1 convexity spaces, exposing a new connection between domains and convex structures. Maruyama [12] also gives closely related dualities for convexity spaces. Jacobs [8] shows a dual adjunction between preframes and algebras of the distribution monad, which are abstract convex structures as well as convexity spaces. We can actually relate the two sorts of abstract convex structures, and thus dualities for them, by several adjunctions and equivalences, though here we do not have space to work out the details.

In the case of sober-type dualities, we first have dual adjunctions for general point-free spaces, which then restrict to dualities (i.e., dual equivalences). In the case of T_1-type dualities, however, we do not have dual adjunctions behind them because we use maximal spectrum Spm rather than prime spectrum Spec. This is the reason why in this paper we have concentrated on the Chu representation of atomistic point-free spaces, rather than point-free spaces in general. We leave it for future work to work out the dual adjunction between $\mathbf{PsSp}_{\mathfrak{X}}$ and $\mathbf{PfSp}_{\mathfrak{X}}$ which restricts to the corresponding sober-type duality.

3 Quantum Symmetries and Closure-Based Coalgebras

We first review the Grothendieck construction for later discussion.

Grothendieck Construction. The Grothendieck construction enables us to glue different categories together into a single category, or turn an indexed category into a fibration. Given a functor $\mathbf{I} : \mathbf{C}^{\mathrm{op}} \to \mathbf{CAT}$, we define a category

$$\int \mathbf{I} : \mathbf{C}^{\mathrm{op}} \to \mathbf{CAT}$$

as follows (**CAT** denotes the category of (small) categories and functors). The objects of $\int \mathbf{I}$ consist of tuples (C, X) where $C \in \mathbf{C}$ and $X \in \mathbf{I}(C)$. An arrow

from (C, X) to (D, Y) in $\int \mathbf{I}$ is defined as a pair (f, g) where $f : D \to C$ and $g : \mathbf{I}(f)(X) \to Y$. Finally, composition of $(f : D \to C, g : \mathbf{I}(f)(X) \to Y) : (C, X) \to (D, Y)$ and $(p : E \to D, q : \mathbf{I}(p)(Y) \to Z) : (D, Y) \to (E, Z)$ is defined as:

$$(f \circ p, q \circ \mathbf{I}(p)(g)) : (C, X) \to (E, Z).$$

Note that the type of $\mathbf{I}(p)(g)$ is $\mathbf{I}(p)(\mathbf{I}(f)(X)) \to \mathbf{I}(p)(Y)$, which in turn equals $\mathbf{I}(f \circ p)(X) \to \mathbf{I}(p)(Y)$. We call $\int \mathbf{I}$ the fibred category constructed from the indexed category $\mathbf{I}$. The obvious forgetful functor from the fibred category $\int \mathbf{I}$ to the base category $\mathbf{C}$ which maps (C, X) to C gives a fibration.

3.1 Born Coalgebras on Closure Spaces

Now, we define an endofunctor $\mathbf{B} : \mathbf{Clos} \to \mathbf{Clos}$ on the category of closure spaces. For a closure space X, let

$$\mathbf{B}(X) := (\{0\} + (0, 1] \times X)^{\mathcal{C}(X)}$$

where $(\{0\}+(0, 1]\times X)^{\mathcal{C}(X)}$ is the product of $\mathcal{C}(X)$-many copies of $\{0\}+(0, 1]\times X$. For a closure-preserving map $f : X \to Y$, we define a map

$$\mathbf{B}(f) : (\{0\} + (0, 1] \times X)^{\mathcal{C}(X)} \to (\{0\} + (0, 1] \times Y)^{\mathcal{C}(Y)}$$

by

$$\mathbf{B}(f)(h)(C) = (id_{\{0\}} + id_{(0,1]} \times f) \circ h \circ f^{-1}(C)$$

where $h \in (\{0\} + (0, 1] \times X)^{\mathcal{C}(X)}$ and $C \in \mathcal{C}(Y)$.

Lemma 14. *For a closure-preserving map $f : X \to Y$, $\mathbf{B}(f)$ is closure-preserving.*

Lemma 15. *Let X, Y, Z be closure spaces. (i) $\mathbf{B}(id_X) = id_{\mathbf{B}(X)}$. (ii) $\mathbf{B}(g \circ f) = \mathbf{B}(g) \circ \mathbf{B}(f)$ for closure-preserving maps $f : X \to Y$ and $g : Y \to Z$.*

Now, we describe primary examples of $\mathbf{B}$-coalgebras, which are of central importance in our investigation.

Example 16 *Given a Hilbert space H, we define a $\mathbf{B}$-coalgebra*

$$(\mathrm{P}(H), \alpha_H : \mathrm{P}(H) \to \mathbf{B}(\mathrm{P}(H)))$$

as follows. Let us define $\alpha_H : \mathrm{P}(H) \to (\{0\} + (0, 1] \times \mathrm{P}(H))^{\mathcal{C}(\mathrm{P}(H))}$ by

$$\alpha_H([\varphi])(S) = \begin{cases} 0 & \text{if } \langle\varphi|P_S\varphi\rangle = 0 \\ (\frac{\langle\varphi|P_S\varphi\rangle}{\langle\varphi|\varphi\rangle}, [P_S\varphi]) & \text{otherwise} \end{cases}$$

where $[\varphi] \in \mathrm{P}(H)$, $S \in \mathrm{L}(H)$ $(\simeq \mathcal{C}(\mathrm{P}(H)))$, and P_S is the projection operator corresponding to S.

The coalgebra $(\mathrm{P}(H), \alpha_H)$ expresses the dynamics of repeated Born-rule-based measurements of a quantum system represented by a Hilbert space H.

As in Abramsky [2], we define the groupoid of quantum symmetries as follows.

Definition 17. QSym *is the category whose objects are projective spaces of Hilbert spaces of dimension greater than* 2 *and whose arrows are semi-unitary maps identified up to a phase factor* $e^{i\theta}$.

Wigner's theorem (or Wigner-Bargmann's theorem) clarifies the physical meaning of **QSym** as follows. Note that "surjections" below are actually bijections, since injectivity follows by the other properties.

Theorem 18. QSym *is equivalent to the category whose objects are projective spaces of Hilbert spaces (i.e., quantum state spaces) and whose arrows are symmetry transformations (i.e., those surjections between projective spaces that preserve transition probabilities* $\frac{|\langle\varphi|\psi\rangle|^2}{|\varphi|^2|\psi|^2}$ *between quantum states* $[\varphi]$ *and* $[\psi]$*).*

Our aim is to establish a purely coalgebraic understanding of **QSym**. We remark that symmetries are of central importance in physics: they are higher laws of conservation of various physical quantities (Nöther's theorem); in quantum mechanics in particular, we can even derive the Schrödinger-equation-based dynamics of quantum systems from a continuous one-parameter group of symmetries (Stone's theorem).

3.2 Quantum Symmetries Are Purely Coalgebraic

For an endofunctor $G : \mathbf{C} \to \mathbf{C}$ on a category $\mathbf{C}$, let $\mathbf{Coalg}(G)$ denote the category of G-coalgebras.

Let us briefly review Abramsky's fibred category $\int \mathbf{F}$ of coalgebras in the following. For a fixed set Q, we define a functor $F^Q : \mathbf{Set} \to \mathbf{Set}$. Given a set X, let $F^Q(X) = (\{0\} + (0,1] \times X)^Q$. The arrow part is then defined canonically.

An indexed category

$$\mathbf{F} : \mathbf{Set}^{\mathrm{op}} \to \mathbf{CAT}$$

is then defined as follows. Given $Q \in \mathbf{Set}$, let $\mathbf{F}(Q) = \mathbf{Coalg}(F^Q)$. For a map $f : Q' \to Q$, we define a functor $\mathbf{F}(f) : \mathbf{Coalg}(F^Q) \to \mathbf{Coalg}(F^{Q'})$ in the following way. Given an object $(X, \alpha : X \to F^Q(X))$ in $\mathbf{Coalg}(F^Q)$, let

$$\mathbf{F}(f)(X, \alpha) = (X, t^f_X \circ \alpha)$$

where $t^f_X : F^Q(X) \to F^{Q'}(X)$ is defined by $t^f_X(g) = g \circ f$. Given an arrow $g : (X, \alpha) \to (Y, \beta)$, let $\mathbf{F}(f)(g) = g : (X, t^f_X \circ \alpha) \to (Y, t^f_Y \circ \beta)$.

As Wigner's theorem above has the assumption of surjectivity, Abramsky [2] requires surjectivity on the first components f of morphisms (f, g) in $\int \mathbf{F}$. Let us denote by $\int \mathbf{F_s}$ the resulting category with the restricted class of morphisms. On the other hand, we require injectivity on the morphisms $f : (X, \alpha) \to (Y, \beta)$ of $\mathbf{Coalg}(\mathbf{B})$, and denote by $\mathbf{Coalg_i}(\mathbf{B})$ the resulting category with the restricted

class of morphisms. The surjectivity/injectivity conditions ensure that **QSym** is not only faithfully but also fully represented in $\int \mathbf{F_s}$ and in $\mathbf{Coalg_i}(\mathbf{B})$.

In the following we observe that $\mathbf{Coalg}(\mathbf{B})$ is much smaller than $\int \mathbf{F}$, but still large enough to encompass the quantum symmetry groupoid **QSym**. To be precise, it shall be shown that $\mathbf{Coalg}(\mathbf{B})$ is a non-full proper subcategory of $\int \mathbf{F}$, and that **QSym** is a full subcategory of $\mathbf{Coalg_i}(\mathbf{B})$.

We then introduce a functor **BF** from $\mathbf{Coalg}(\mathbf{B})$ to $\int \mathbf{F}$, which will turn out to be a non-full embedding of categories.

Definition 19. *The object part of* $\mathbf{BF} : \mathbf{Coalg}(\mathbf{B}) \to \int \mathbf{F}$ *is defined by*

$$\mathbf{BF}(X, \alpha : X \to \mathbf{B}(X)) = (\mathcal{C}(X), (X, \alpha) \in \mathbf{Coalg}(F^{\mathcal{C}(X)})).$$

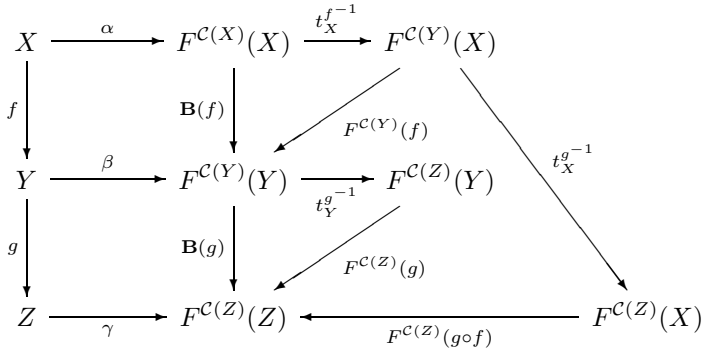
The arrow part of $\mathbf{BF} : \mathbf{Coalg}(\mathbf{B}) \to \int \mathbf{F}$ *is defined by*

$$\mathbf{BF}(f : (X, \alpha) \to (Y, \beta)) = (f^{-1} : \mathcal{C}(Y) \to \mathcal{C}(X), \tilde{f} : \mathbf{F}(f^{-1})(X, \alpha) \to (Y, \beta))$$

where $\tilde{f}$ *has the same underlying function as* f *(i.e.,* $\tilde{f}(x) = f(x)$ *for any* $x \in X$*; thus, the difference only lies in their types).*

In order to justify the definition above, we have to verify that $\tilde{f}$ is actually a morphism in $\mathbf{Colag}(F^{\mathcal{C}(Y)})$.

The commutative diagram below would be useful to understand what is going on in the definition above and the two lemmata below.



where α, β, γ are **B**-coalgebras, and f, g are morphisms of **B**-coalgebras.

Lemma 20. $\tilde{f} : \mathbf{F}(f^{-1})(X, \alpha) \to (Y, \beta)$ *is an arrow in* $\mathbf{Colag}(F^{\mathcal{C}(Y)})$.

Lemma 21. *(i)* $\mathbf{BF}(id_{(X,\alpha)}) = id_{\mathbf{BF}(X,\alpha)}$. *(ii) For* $f : (X, \alpha) \to (Y, \beta)$ *and* $g : (Y, \beta) \to (Z, \gamma)$ *in* $\mathbf{Coalg}(\mathbf{B})$, $\mathbf{BF}(g \circ f) = \mathbf{BF}(g) \circ \mathbf{BF}(f)$ *where the latter composition is that in* $\int \mathbf{F}$.

Proposition 22. $\mathbf{Coalg}(\mathbf{B})$ *can be embedded into* $\int \mathbf{F}$ *via the functor* **BF**. *This is not a full embedding (i.e.,* **BF** *is not full).*

The non-fullness of **BF** implies that $\mathbf{Coalg}(\mathbf{B})$ is a smaller category than $\int \mathbf{F}$ with respect to arrows as well as objects.

We now introduce a functor **SC** from **QSym** to $\mathbf{Coalg_i}(\mathbf{B})$, which will turn out to be a full embedding of categories.

Definition 23. *The object part of* $\mathbf{SC} : \mathbf{QSym} \to \mathbf{Coalg_i}(\mathbf{B})$ *is defined by*

$$\mathbf{SC}(\mathrm{P}(H)) = (\mathrm{P}(H), \alpha_H).$$

The arrow part of $\mathbf{SC} : \mathbf{QSym} \to \mathbf{Coalg_i}(\mathbf{B})$ *is defined by*

$$\mathbf{SC}(U) = U : (\mathrm{P}(H), \alpha_H) \to (\mathrm{P}(H'), \alpha_{H'})$$

where $U : \mathrm{P}(H) \to \mathrm{P}(H')$ *is a semi-unitary map from* H *to* H' *(up to a phase).*

Lemma 24. $\mathbf{SC}(U)$ *is a morphism of* $\mathbf{B}$*-coalgebras.*

W finally obtain the purely coalgebraic representation of quantum symmetries **QSym** via the non-fibred, single sort of coalgebra category $\mathbf{Coalg_i}(\mathbf{B})$ based upon closure spaces.

Theorem 25. *The quantum symmetry groupoid* **QSym** *can be fully embedded into the purely coalgebraic category* $\mathbf{Coalg_i}(\mathbf{B} : \mathbf{Clos} \to \mathbf{Clos})$.

Our closure-based coalgebraic approach to representation of quantum systems would allow us to develop "coalgebraic quantum logic" utilising existing work on coalgebraic logic over (duality between) general concrete categories (see, e.g., Kurz [11] or Klin [10]); this is left for future work.

References

1. Abramsky, S.: Coalgebras, Chu spaces, and representations of physical systems. In: Proc. of LICS 2010, pp. 411–420 (2010)
2. Abramsky, S.: Big toy models: representing physical systems as Chu spaces. Synthese 186, 697–718 (2012)
3. Adámek, J., Herrlich, H., Strecker, G.E.: Abstract and Concrete Categories. John Wiley and Sons, Inc. (1990)
4. Bonsangue, M.M., Jacobs, B., Kok, J.N.: Duality beyond sober spaces: topological spaces and observation frames. Theor. Comput. Sci. 151, 79–124 (1995)
5. Coekce, B., Moore, D.J.: Operational Galois adjunctions. In: Current Research in Operational Quantum Logic, pp. 195–218 (2000)
6. Gierz, G., Hofmann, K.H., Keimel, K., Lawson, J.D., Mislove, M.W., Scott, D.S.: Continuous Lattices and Domains. Cambridge University Press (2003)
7. Hartshorne, R.: Algebraic Geometry. Springer (1977)
8. Jacobs, B.: Convexity, duality and effects. In: Calude, C.S., Sassone, V. (eds.) TCS 2010. IFIP AICT, vol. 323, pp. 1–19. Springer, Heidelberg (2010)
9. Johnstone, P.T.: Stone Spaces. Cambridge University Press
10. Klin, B.: Coalgebraic modal logic beyond sets. Electr. Notes Theor. Comput. Sci. 173, 177–201 (2007)
11. Kurz, A.: Coalgebras and their logics. ACM SIGACT News 37 (2006)
12. Maruyama, Y.: Fundamental results for pointfree convex geometry. Ann. Pure Appl. Logic 161, 1486–1501 (2010)
13. Maruyama, Y.: Natural duality, modality, and coalgebra. Journal of Pure and Applied Algebra, 565–580 (2012)

14. Moore, D.J.: On state spaces and property lattices. Stud. Hist. Phil. Mod. Phys. 30, 61–83 (1999)
15. Porst, H.E., Tholen, W.: Concrete dualities. In: Category Theory at Work, pp. 111–136 (1991)
16. Pratt, V.R.: The Stone Gamut: a coordinatization of mathematics. In: Proc. of LICS 1995, pp. 444–454 (1995)
17. Rutten, J.J.M.M.: Universal coalgebra: a theory of systems. Theor. Comput. Sci. 249, 3–80 (2000)
18. van de Vel, M.L.J.: Theory of Convex Structures. North-Holland (1993)

Noninterfering Schedulers

When Possibilistic Noninterference Implies Probabilistic Noninterference

Andrei Popescu, Johannes Hölzl, and Tobias Nipkow

Technische Universität München

Abstract. We develop a framework for expressing and analyzing the behavior of probabilistic schedulers. There, we define noninterfering schedulers by a probabilistic interpretation of Goguen and Meseguer's seminal notion of noninterference. Noninterfering schedulers are proved to be safe in the following sense: if a multi-threaded program is possibilistically noninterfering, then it is also probabilistically noninterfering when run under this scheduler.

1 Introduction

Noninterference is an important and well-studied formal property modeling *confidentiality*. It was introduced by Goguen and Meseguer (henceforth abbreviated G&M) in the context of deterministic multi-user systems having essentially the following meaning [5, p.11]: "One group of users is noninterfering with another group of users if what the first group does has no effect on what the second group of users can see."

In the context of confidentiality in a language-based setting [16], a quite different notion, usually also termed as noninterference, emerged in work by Volpano et. al. [23]: Assuming the program memory is separated into a *low*, or public, part, which an attacker is able to observe, and a *high*, or private, part, hidden to the attacker, a *sequential* program satisfies noninterference if, upon running it, the high part of the initial memory does not affect the low part of the resulting memory.

Of course, many systems for which confidentiality is important are *concurrent*, such as Internet servers or operating systems. To cope with concurrency, the above language-based notion of noninterference has been generalized in various ways. A major line of work focuses on *possibilistic noninterference*, which roughly states that if an execution allowing certain observations by the attacker is possible, then another execution for which these observations are infirmed is also possible. For this notion, powerful and/or compositional analysis methods have been devised [1, 3, 8, 20]. The downside of possibilistic noninterference is vulnerability under *probabilistic attacks* by running the program multiple times and gathering statistical information and *refinement attacks* via knowledge of the thread scheduling. For example, consider the following program consisting of two threads running in parallel under a uniform probabilistic scheduler [18]:

- $\mathsf{while}\ h > 0\ \mathsf{do}\ \{h := h - 1\}\ ;\ l := 2$
- $l := 1$

Then, probability to execute $l := 2$ after $l := 1$, i.e., to obtain 2 for the final l, depends on the initial value of h, making the latter inferable from the distribution of the final l.

R. Heckel and S. Milius (Eds.): CALCO 2013, LNCS 8089, pp. 236–252, 2013.

These problems have been addressed by introducing several (overlapping) notions of *probabilistic* [9,18,19] and *scheduler-independent* [9,17,24] noninterference and means to enforce them. Proposed scheduler-independent solutions (probabilistic or not) insure confidentiality in the presence of any scheduler [16, 17, 20] or a large class of schedulers [4, 9, 13], but suffer from various limitations: lack of coping with dynamic thread creation [4], too harsh requirements on individual threads (strong security) [17, 20], too weak confidentiality guarantees on the overall concurrent system [9], the reliance on expensive or not always feasible conditions such as race freedom [24] or termination [9], or non-standard thread-level security primitives [13].

This paper presents a way to alleviate these limitations in a scheduler-dedicated framework. Its main contributions are:

- A framework for analyzing schedulers independently from the concrete operational semantics of threads.
- A notion of noninterfering scheduler obtained by a novel reading, in a probabilistic key, of G&M's seminal notion.
- A result inferring probabilistic noninterference from possibilistic noninterference under the assumption of a noninterfering scheduler (for suitable notions of possibilistic and probabilistic noninterference).

This result captures a large class of schedulers, covers dynamic thread creation, allows timing of thread execution to depend on high data, guarantees a strong security property on the thread system, and does not rely on undecidable properties of the multi-threaded program or special security primitives. Our scheduler noninterference, importing insights from system-based noninterference to language-based noninterference, is a step toward better understanding the complex relationship between these two worlds [7].

We start by introducing the framework for schedulers (§2), carefully factoring in *all* and *only* the information relevant to scheduling. Thus, in order to have fine control over the scheduling policy including dynamic thread creation, we keep an order on thread IDs that indicate who spawned who. On the other hand, for studying the behavior of a scheduler we do not employ concrete thread pools with state-based semantics for threads—instead, we consider *execution scenarios*, i.e., possibilistic interleavings of threads IDs to which the scheduler casts probabilities.

Operational semantics of multi-threaded programs (§3) is separated in two: The *possibilistic semantics* is the usual nondeterministic interleaving semantics; in particular, it yields an execution scenario for each pair (program, initial state). A *probabilistic semantics* is obtained by blending in a scheduler with the execution scenario.

Then we move to discussing noninterference (§4). For defining scheduler noninterference, we identify two groups of users à la G&M: the threads that are *visible*, i.e., will eventually affect the observable part of the system during their execution, and the others, the *invisible* ones. The user's *actions* are, as expected, the very steps taken by the threads. The *observations*, however, require a nonstandard interpretation: Given a visible thread v and letting I denote the collection of invisible threads, the observation of v is the "exit probability" of v through I, i.e., the probability of the system taking zero or more I-steps followed by a v-step. We call the scheduler noninterfering if the observation of each visible thread v is independent of the actions of the invisible threads I,

i.e., the exit probability of v through I is the same as the probability of taking v provided I were completely inexistent (including from the execution history).

For *possibilistic* noninterference of programs, we adopt a compositional notion introduced in [9], a weakening of strong security [17] allowing the execution time to depend on secrets. In fact, our approach as a whole disregards execution time. We take *probabilistic* noninterference to be the notion introduced by Smith [18]: Any two executions that differ only on secret information traverse the same sequence of attacker observations with the same probability—this seems to be the strongest notion of probabilistic noninterference that ignores timing channels.

Further details on the constructions and results from this paper, including more substantial proof sketches, can be found in the technical report [10], which is an identical copy of the paper save for an appendix with additional material.

2 Framework for Schedulers

This section introduces the key component of our approach: a framework for studying schedulers in isolation from the concrete (state-based) operational semantics.

In the noninterference literature (e.g., [9, 17, 18]), the thread IDs manipulated by schedulers are typically handled implicitly, as the numeric indexes (positions) of the threads in the pool represented as a list. However, here we endow thread IDs with more structure, able to store information about the parent thread and the order in which the current threads have been spawned (§2.1). We introduce *histories*, i.e., sequences of thread IDs taken so far during the execution, and *rich histories* obtained from augmenting the histories with information about the threads that were available at each point in history—these enriched structures offer useful information concerning the thread waiting time (§2.2). Schedulers are defined as operating on rich histories (§2.3). In order to study scheduler noninterfernce in isolation from a concrete operational semantics, we single out the aspect of thread semantics relevant for the scheduler's behavior: *execution scenarios*, as trees of thread ID interleavings (§2.4). Given an execution scenario, a scheduler induces a Markov chain structure on histories, offering a quantitative interpretation of temporal logic formulas (§2.5) useful later for defining noninterference.

2.1 Thread IDs

We let i, j, k, l range over natural numbers. The *thread IDs*, ranged over by m, n, p, q, will be elements of the set $\mathbf{threadID} = \mathbf{nat}^*$, of words (i.e., finite sequences) of natural numbers. The empty sequence ε will represent the main thread's ID. We write $m{\cdot}n$ for the concatenation of m and n. As usual, we identify single numbers k with singleton words, and thus $k{\cdot}m$ and $m{\cdot}k$ represent the words obtained by pre-appending and post-appending k to m, respectively.

For all $m \in \mathbf{threadID}$, we define the set of IDs that m *may spawn*, $\mathsf{maySp}\ m$, as $\{m{\cdot}k \mid k \in \mathbf{nat}\}$. The full reading of "$n \in \mathsf{maySp}\ m$" is the following: "if m is the ID of a given thread, then n is valid as ID for a thread the given thread may spawn (in the future)". Note that $\forall n.\ \varepsilon \notin \mathsf{maySp}\ n$, i.e., no thread may spawn the main thread.

We also define, for each $m \in \mathbf{threadID}$, the following order $<_m$ on $\mathsf{maySp}\ m$, called the *m-issuing order*: $m{\cdot}k <_m m{\cdot}j$ iff $k < j$. The reading of "$p <_m q$" is "the thread ID p

should be generated before q is (in any potential execution)". For example, it holds that $2{\cdot}1 \in \mathsf{maySp}\ 2$ and $2{\cdot}1{\cdot}1 <_{2{\cdot}1} 2{\cdot}1{\cdot}2$.

The "may spawn" operator and the issuing orders will be means to inform the scheduler about who spawned who and about the order in which spawning happened. In our informal explanations, we shall loosely identify threads with thread IDs.

2.2 Histories

Our schedulers will depend on execution histories indicated as lists of thread IDs. Since on the other hand thread IDs are themselves modeled as lists (sequences), to avoid confusion we use a different notation for lists of threads.

Namely, we let **hist**, the set of *(execution) histories*, ranged over ml, nl, pl, ql, consist of thread ID lists. $[m_0, \ldots, m_{k-1}]$ denotes the history consisting of the indicated thread IDs in the indicated order. $[]$ is the empty history and $[m]$ is a singleton history. $ml \,\#\, nl$ denotes the concatenation of histories ml and nl, and we write $ml \,\#\, n$ and $n \,\#\, ml$ instead of $ml \,\#\, [n]$ and $[n] \,\#\, ml$, respectively. If $ml = [m_0, \ldots, m_{k-1}]$, $ml\langle..i\rangle$ is the subhistory of ml containing the first i elements, $[m_0, \ldots, m_{i-1}]$; thus, $ml\langle..0\rangle = []$ and $ml\langle..k\rangle = ml$.

Given $n \in \mathbf{threadID}$, $N \subseteq \mathsf{maySp}\ n$ and $M \subseteq \mathbf{threadID}$, M is called an *initial fragment of N w.r.t.* $<_n$ if $M \subseteq N$ and $\forall m \in M.\ \forall m' \in N \setminus M.\ m <_n m'$.

We shall be interested in the relationship between execution histories and the sets of available threads at each point in such histories. We let Ml range over lists of finite sets of thread IDs. A pair (ml, Ml) where $ml = [m_0, \ldots, m_{k-1}]$ and $Ml = [M_0, \ldots, M_k]$ (thus having $\mathsf{length}\ Ml = \mathsf{length}\ ml + 1$) is said to be:

- *start-consistent*, if $M_0 = \{\varepsilon\}$;
- *step-consistent*, if $\forall i < k.\ m_i \in M_i$;
- *termination-consistent*, if $\forall i < k.\ M_i \setminus M_{i+1} \subseteq \{m_i\}$;
- *spawn-consistent*, if $\forall i < k.\ M_{i+1} \setminus M_i$ is an initial fragment of $\mathsf{maySp}\ m_i \setminus (M_0 \cup \ldots \cup M_i)$ w.r.t. $<_{m_i}$.

The above conditions describe the correct interplay between the threads available in the pool at each moment (represented by Ml) and single execution steps taken by the threads (represented by ml). More precisely, we assume that, at moment i, M_i are the available threads and m_i takes a step, yielding the available threads M_{i+1}.

Start consistency: Execution starts with the main thread alone in the thread pool.

Step consistency: Only an available thread can take a step.

Termination consistency: Upon a step taken by thread m_i, the resulted thread pool contains all threads except perhaps m_i (if terminated).

Spawn consistency: Upon a step taken by thread m_i, all newly appearing threads in the pool get IDs that m_i may spawn and, moreover, they get the smallest such IDs that are *fresh*, in the sense of not having been assigned before.

Note that start consistency and step consistency imply that $m_0 = \varepsilon$. A pair (ml, Ml) is called a *rich history* if it is start-, step-, termination-, and spawn- consistent. We let **rhist** denote the set of rich histories.

Rich histories $(ml = [m_0, \ldots, m_{k-1}], Ml = [M_0, \ldots, M_k])$ contain enough information to determine various moments in the life span of threads:

- The set of *current threads*, $\mathsf{Cur}\ Ml$, is the last element in this list, M_k.
- Given $n \in \mathsf{Cur}\ Ml \setminus \{\varepsilon\}$, the *moment when n appeared*, $\mathsf{app}_{Ml}\ n$, is the smallest i such that $n \in M_{i+1}$.
- Given $n \in \{m_0, \ldots, m_{k-1}\}$, the *moment when n was last taken (executed)*, $\mathsf{ltaken}_{ml}\ n$, is the greatest $i < k$ such that $n = m_i$.
- Given $n \in \mathsf{Cur}\ Ml$, the *moment when n was last touched*, $\mathsf{ltouched}_{ml,Ml}\ n$, is: either $\mathsf{ltaken}_{ml}\ n$, if $n \in \{m_0, \ldots, m_{k-1}\}$; or $\mathsf{app}_{Ml}\ n$, otherwise.
- Given $n \in \mathsf{Cur}\ Ml$, the *waiting time for n*, $\mathsf{wait}_{ml,Ml}\ n$, is $k - 1 - \mathsf{ltouched}_{ml}\ n$.

Note that, if both $\mathsf{app}_{Ml}\ n$ and $\mathsf{ltaken}_{ml}\ n$ are defined, i.e., if $n \in \mathsf{Cur}\ Ml \setminus \{\varepsilon\} \cap \{m_0, \ldots, m_{k-1}\}$, then, by step-consistency, $\mathsf{app}_{Ml}\ n < \mathsf{ltaken}_{ml}\ n$—this justifies our definition for $\mathsf{ltouched}_{ml,Ml}\ n$.

2.3 Schedulers

A *scheduler* is a family of functions $(sch_{ml,Ml} : \mathsf{Cur}\ Ml \to \mathbb{R})_{ml,Ml}$, where (ml, Ml) ranges over rich histories, such that $\forall m \in \mathsf{Cur}\ Ml.\ sch_{ml,Ml}\ m \geq 0$ and $\sum_{m \in \mathsf{Cur}\ Ml} sch_{ml,Ml}\ m = 1$. Thus, given a rich history (ml, Ml), a scheduler defines a probability distribution $sch_{ml,Ml}$ on the currently available threads $\mathsf{Cur}\ Ml$. Next we give two standard examples. (See [10, §A] for several others.)

Uniform Scheduler. usch assigns all currently available threads equal (history oblivious) probability: $\mathsf{usch}_{ml,Ml}\ m = 1/|\mathsf{Cur}\ Ml|$. Uniform scheduling is the underlying assumption in work by Smith and Volpano on probabilistic noninterference [18, 19, 22].

Round Robin Scheduler. Given a number j, the round robin scheduler with j step quotas, rsch^j, always schedules with probability 1 the first thread in the queue for j consecutive steps, where threads are ordered in a queue according to their waiting time.

Given (ml, Ml), we define the following *queuing order* on M: $n <_{ml,Ml} n'$ iff

- either $\mathsf{wait}_{ml,Ml}\ n < \mathsf{wait}_{ml,Ml}\ n'$,
- or $\mathsf{wait}_{ml,Ml}\ n = \mathsf{wait}_{ml,Ml}\ n'$ and $n' \in \mathsf{maySp}\ n$,
- or else $n, n' \in \mathsf{maySp}\ p$ and $n' <_p n$ for some p.

$<_{ml,Ml}$ organizes the current thread pool M as a queue based on waiting times, resolving same-waiting-time conflicts as follows: a spawned thread has priority over its parent, two threads spawned at the same time are discriminated by the issuing order. The first (maximum) in this waiting queue, $\max_{ml,Ml}\ M$, is in the set of threads with highest waiting time and, among these, is the smallest w.r.t. the "may spawn" and issuing orders.

For any history ml, we let $\$(ml)$ be the number of trailing occurrences of its last thread, i.e., the largest number k such that ml has the form $nl \# m^k$, where m^k consists of k repetitions of m. We define rsch^j, the *j-step round robin scheduler*, as follows, for all $(ml, Ml) \in \mathbf{rhist}$ and $p \in M = \mathsf{Cur}\ Ml$:

– If $ml = []$, then necessarily $Ml = \{\varepsilon\}$, $M = \{\varepsilon\}$ and $p = \varepsilon$. We put $\mathsf{rsch}^j_{ml,Ml}\ p = 1$.

– If ml has the form $nl \# m$, then we define

$$\mathsf{rsch}^j_{ml,Ml}\, p = \begin{cases} 1, \text{ if } \$(ml) < j \wedge p = m, & \text{(last scheduled thread } m \text{ still has quota)} \\ 1, \text{ if } \$(ml) \geq j \wedge p = \max_{ml,Ml} M, & (m \text{ finished its quota, } p \text{ comes next}) \\ 1, \text{ if } m \notin M \wedge p = \max_{ml,Ml} M, & (m \text{ has terminated, } p \text{ comes next}) \\ 0, \text{ otherwise.} & (p \text{ neither current, nor next to be scheduled}) \end{cases}$$

Previous work on concurrent noninterference [9, 12, 14, 15, 17] considers round robin schedulers almost equivalent to our rsch^j, except for the policy of placing in the pool the newly spawned threads. Namely, while defining the operational semantics of the thread pools modeled as lists of threads, the newly spawned threads are inserted in the list *after* the parent thread. This, together with the policy of the scheduler tape traversing the thread pool from left to right, makes the scheduler non-starvation-free—e.g., if m spawns in one step an identical copy of itself, that copy would be scheduled immediately after m, and thus m and its clones would monopolize execution. Our definition based on the waiting time avoids this problem. Of course, the problem is also solvable by changing the operational semantics to traverse the thread list from right to left instead. However, this solution reveals a limitation of approaches that hardwire in the thread-pool operational semantics the policy for placing new threads: the need for global changes in order to accommodate desired scheduler properties. By contrast, our approach packs up the whole scheduler behavior in the definition of the scheduler alone.

2.4 Execution Scenarios

Although a scheduler depends on rich histories which are essentially *linear* structures, its behavior is better comprehended through what we call execution scenarios, tree-like structures that capture the *branching* of thread interleaving. Given any set H of histories and given $ml = [m_0, \ldots, m_{k-1}] \in H$ such that all its prefixes are also in H:

– Let $\mathsf{Avail}^H\, ml$, the set of thread IDs *available in H at point ml*, be $\{m \in \mathbf{threadID} \mid ml \# m \in H\}$;
– Let $\mathsf{Havail}^H\, ml$, the list of sets of thread IDs *available in H all throughout history ml*, be $[\mathsf{Avail}^H\, ml\langle..0\rangle, \ldots, \mathsf{Avail}^H\, ml\langle..k\rangle]$.
– Given $m \in \mathsf{Avail}^H\, ml$, let $\mathsf{spawns}^H_{ml}\, m$, the *set of threads spawned by one m-step at history ml*, be $\mathsf{Avail}^H\, (ml \# m) \setminus \mathsf{Avail}^H\, ml$.

An *(execution) scenario* is a set Sc of histories such that the following properties hold, where $\preceq$ is the prefix order on lists:

– Prefix Closure: $\forall ml\, nl.\ nl \in Sc \wedge ml \preceq nl \Longrightarrow ml \in Sc$.
– Finite Branching: $\forall ml \in Sc.\ \mathsf{Avail}^{Sc}\, ml$ is finite.
– Consistency: $\forall ml \in Sc.\ (ml, \mathsf{Havail}^{Sc}\, ml) \in \mathbf{rhist}$.
– Boundedness: $\exists k.\ \forall ml \in Sc.\ \forall m \in \mathsf{Avail}^{Sc}\, ml.\ |\mathsf{spawns}^{Sc}_{ml}\, m| \leq k$.

Thus, a scenario is required to form a finitely branching tree for which all finite paths are rich histories and there exists a bound on the number of threads spawned concurrently in one single step. The best way to picture a scenario is as a labeled tree, where the nodes are histories ml (with $[]$ as the root), and the edges coming out of each ml are labeled with the elements of $\mathsf{Avail}^{Sc}\, ml$. For example, if we ignore the circled numbers for now,

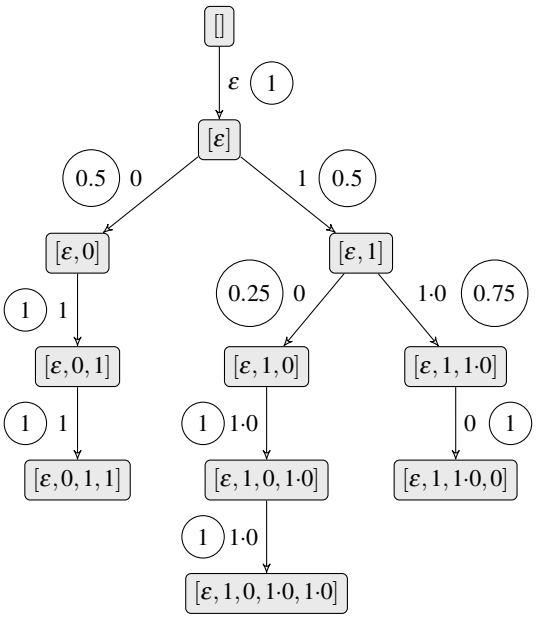


Fig. 1. A scenario with probabilities attached

Fig. 1 shows the finite scenario $Sc = \{[], [\varepsilon], [\varepsilon,0], [\varepsilon,1], [\varepsilon,0,1], [\varepsilon,1,0], [\varepsilon,1,1{\cdot}0], [\varepsilon,0,1,1], [\varepsilon,1,0,1{\cdot}0], [\varepsilon,1,1{\cdot}0,0], [\varepsilon,1,0,1{\cdot}0,1{\cdot}0]\}$. In this scenario, the following happen (among other things): at history $[]$, ε takes one step and terminates, with spawning two threads, 0 and 1 (hence $\mathsf{spawns}^{Sc}_{[]}\,\varepsilon = \{0,1\}$)—this can be seen from the branchings of $[]$ and $[\varepsilon]$: ε is available at history $[]$, while 0 and 1 are available at the successor history $[\varepsilon]$; at history $[\varepsilon]$, after taking one step, 1 terminates, spawning a new thread $1{\cdot}0$; at history $[\varepsilon,0]$, 1 takes two steps and terminates, without spawning any threads.

Note that, in accordance with termination consistency, the described scenario never abandons execution, but proceeds until termination is plausible. E.g., from its appearance (at history $[\varepsilon]$), on any path thread 0 is continuously available before it is taken.

2.5 Scheduler-Induced Probabilities on Scenarios

Given a scenario Sc, a scheduler sch assigns probabilities to branches in Sc—at history ml, the branch $m \in \mathsf{Avail}^{Sc}\,ml$ receives probability $sch_{ml,\,\mathrm{Havail}^{Sc}\,ml}\,m$—and then to finite paths in Sc as the product of probabilities of the branches taken along the path. Fig. 1 shows in circles a possible such assignment of probabilities to a scenario, where, e.g., the history path $[\varepsilon,1,1{\cdot}0]$ has probability $1 * 0.5 * 0.75 = 0.375$.

More generally, let $pl \in Sc$. We let Trace^{Sc}_{pl} be the set of (Sc,pl)-*traces*, which are maximal (finite or infinite) sequences mt such that $pl \,\#\, ml \in Sc$ for all finite prefixes ml of mt. Then we can identify each ml such that $pl \,\#\, ml \in Sc$ with a "basic event" $\mathsf{Bev}^{Sc}_{pl,ml}$ consisting of all (Sc,pl)-traces that start with ml, i.e., have ml as a prefix; we

thus postulate that $\mathsf{Bev}^{Sc}_{pl,ml}$ has the probability of ml when taken in history pl. E.g., in Fig. 1, $\mathsf{Bev}^{Sc}_{[\varepsilon],[1]}$ consists of $\{[1,0,1{\cdot}0,1{\cdot}0],\ [1,1{\cdot}0,0]\}$ and has probability 0.5.

By standard probability theory [6], one can now assign probabilities $\mathsf{P}^{Sc,sch}_{pl}\ Mt$ to certain measurable sets Mt of (Sc,pl)-traces, namely, to those in the smallest collection of subsets of Trace^{Sc}_{pl} that is closed under countable union and complement and contains every $\mathsf{Bev}^{Sc}_{pl,ml}$ for which $pl\,\#\,ml \in Sc$. The *Markov chain induced by sch on Sc*, Mc^{sch}_{Sc}, is the family $(\mathsf{Trace}^{Sc}_{pl}, \mathsf{P}^{Sc,sch}_{pl})_{pl \in Sc}$.

The sets of traces describable in linear temporal logic (LTL) are measurable [21]. Thus, to each LTL formula φ, for each history point $pl \in Sc$, we can speak of the (Sc,sch)-*probability of* φ, written $\mathsf{P}^{Sc,sch}_{pl}\ \varphi$ and defined as the probability of the set of (Sc,pl)-traces satisfying φ. Of particular importance for us will be the following LTL formulas and connectives, where U : **hist** $\to$ **threadID** $\to$ **bool**, $n \in$ **threadID** and φ and χ are any LTL formulas:

$\mathsf{Takes}\ U$, satisfied by a (Sc,pl)-trace iff that trace takes as first step an element m such that of $U\ pl\ m$ holds.

$\mathsf{Ev}\ \varphi$, satisfied by a trace iff φ eventually holds on some point on that trace.

$\mathsf{Alw}\ \varphi$, satisfied by a trace iff φ always holds (on every point) on that trace.

$\varphi\ \mathsf{Until}\ \chi$, satisfied by a trace iff φ holds on every point on some finite initial fragment of that trace, and χ holds immediately after. (This is the LTL "strong until".)

If U simply tests for equality to a fixed thread n, i.e., $\forall ml\ m.\ U\ ml\ m \iff m = n$, we write $\mathsf{Takes}\ n$ instead of $\mathsf{Takes}\ U$. Note that $\mathsf{Ev}\ (\mathsf{Takes}\ n)$ is satisfied by a trace iff that trace contains n, and $(\mathsf{Takes}\ U)\ \mathsf{Until}\ (\mathsf{Takes}\ n)$ is satisfied by a trace iff that trace takes for a while steps for which U holds, and eventually it takes n.

3 Operational Semantics of Programs

Next we introduce the state-based small-step semantics, both possibilistic and probabilistic, for shared-memory multi-threaded programs featuring dynamic thread creation.

3.1 Possibilistic Semantics

Let **state**, ranged over by s,t, be an unspecified set of memory states. We assume that the individual threads are commands $c,d \in$ **cmd** with a semantics given by a transition relation $c \xrightarrow{s} (\gamma,[c_1,\ldots,c_l],s')$, where γ is either $\bot$ or a command c', having the following interpretation: in state s, c takes one step, spawning threads $c_1,\ldots,c_l$, changing the state to s', and: terminating, provided $\gamma = \bot$, or yielding the continuation c', provided γ is a command c'. We assume the transition relation to be total and deterministic, i.e., for all c and s there exists a unique pair $(\gamma,[c_1,\ldots,c_l])$ such that $c \xrightarrow{s} (\gamma,[c_1,\ldots,c_l])$. Also, we assume that each command c is spawn-bounded, in that there exists k (depending on c) such that the number of threads spawned *in one single step* by c or any of its continuations or spawned threads during execution is $\leq k$—this is a reasonable assumption for programs written in a concurrent language, where k can be determined by inspecting the syntax. (Spawn-boundedness has an obvious coinductive definition that we omit.)

A *(runtime) configuration* is a tuple $cf = (ml, Ml, thr, s)$ such that (ml, Ml) is a rich history and $thr : \mathsf{Cur}\ Ml \to \mathbf{cmd}$. (ml, Ml) indicates the execution so far, thr the assignment of commands to thread IDs, s the current memory state. We define a labeled transition relation on configurations: $(ml, Ml = [M_0, \ldots, M_k], thr, s) \xrightarrow{m} (ml', Ml', thr', s')$ iff $m \in M_k$ and the following hold, assuming $thr\, m \xrightarrow{s} (\gamma, [c_1, \ldots, c_l], s')$ and letting $p_1 <_m \ldots <_m p_l$ be the first l smallest thread IDs in $\mathsf{maySp}\, m \setminus (M_0 \cup \ldots \cup M_k)$ w.r.t. $<_m$:

- $ml' = ml \# m$.
- $Ml' = Ml \# M'$, where $M' = \begin{cases} M_k \setminus \{m\} \cup \{p_1, \ldots, p_l\}, & \text{if } \gamma = \bot, \\ M_k \cup \{p_1, \ldots, p_l\}, & \text{otherwise.} \end{cases}$
- thr' behaves like thr on elements of $M' \cap M_k$ and additionally sends each p_i to c_i.

The above is the expected one-step semantics of configurations: any currently available thread may take a (possibly terminating) step, spawning 0 or more new threads that are assigned the smallest available thread IDs, and affecting the state; in case of termination, the thread is removed from the pool.

We define $cf \xrightarrow{[m_1, \ldots, m_k]} cf'$ to mean that there exist $cf_0, \ldots, cf_{k-1}$ such that $cf_0 = cf$, $cf_{k-1} = cf'$, and $cf_i \xrightarrow{m_{i+1}} cf_{i+1}$ for all $i < k$.

3.2 From Possibilistic to Probabilistic Semantics, via Schedulers

Given c and s, let the *initial configuration of* (c, s), $\mathsf{init}\ (c, s)$, be $([], [\{\varepsilon\}], \varepsilon \mapsto c, s)$. Thus, in $\mathsf{init}\ (c, s)$, c is the single (main) thread and s the current state; during execution, c may of course spawn other threads that will populate the configuration. We define $\mathsf{Sc}_{c,s}$, the *scenario of* (c, s), to be $\{ml.\ \exists cf.\ \mathsf{init}\ (c, s) \xrightarrow{ml} cf'\}$—that Sc is indeed a scenario follows immediately from the definition of configuration transitions.

Note that, for each $ml \in \mathsf{Sc}_{c,s}$, there exists precisely one $cf = (ml, Ml, thr, s)$ such that $\mathsf{init}\ (c, s) \xrightarrow{ml} cf$—we write $\mathsf{config}_{c,s}\ ml$ for this cf. Thus, the pair $(\mathsf{Sc}_{c,s}, \mathsf{config}_{c,s})$ constitutes an alternative description of the *possibilistic* semantics of (c, s) (including complete information about thread spawning and termination). If we also factor in the Markov chain induced by *sch* on $\mathsf{Sc}_{c,s}$, we obtain a proper notion of *probabilistic* semantics of (c, s) as the triple $(\mathsf{Sc}_{c,s}, \mathsf{config}_{c,s}, \mathsf{Mc}^{sch}_{c,s})$, where we write $\mathsf{Mc}^{sch}_{c,s}$ instead of $\mathsf{Mc}^{sch}_{\mathsf{Sc}_{c,s}}$. We shall also write $\mathsf{Trace}^{c,s}$ and $\mathsf{P}^{sch,c,s}$ instead of $\mathsf{Trace}^{\mathsf{Sc}_{c,s}}$ and $\mathsf{P}^{\mathsf{Sc}_{c,s},sch}$.

4 Noninterference

Here we present our main security result: a notion of noninterfering scheduler that ensures lifting of possibilistic noninterference to probabilistic noninterference. All throughout this section, we fix a scheduler *sch* and a domain **odom** of observables.

4.1 Noninterfering Schedulers

An *observation-augmented scenario* (*OA-scenario*) is a pair (Sc, obs), where $obs : Sc \to \mathbf{odom}$. Let (Sc, obs) be an OA-scenario. A thread n is called *visible* at a certain history

if it is available and, at some point in the future, n will affect the observables, either directly or indirectly via a spawned thread n', or via a thread spawned by n', etc. Formally, we define inductively the sets $\mathsf{visAvail}^{Sc,obs}\, ml$ of *visible threads available at ml*:

$$\frac{n \in \mathsf{Avail}^{Sc}\, ml \quad obs\,(ml \,\#\, n) \neq obs\, ml}{n \in \mathsf{visAvail}^{Sc,obs}\, ml} \qquad \frac{m,n \in \mathsf{Avail}^{Sc}\, ml \quad n \in \mathsf{visAvail}^{Sc,obs}\,(ml \# m)}{n \in \mathsf{visAvail}^{Sc,obs}\, ml}$$

$$\frac{n \in \mathsf{Avail}^{Sc,obs}\, ml \quad n' \in \mathsf{spawns}^{Sc}_{ml}\, n \quad n' \in \mathsf{visAvail}^{Sc,obs}\,(ml \,\#\, n)}{n \in \mathsf{visAvail}^{Sc,obs}\, ml}$$

An available thread m is called *invisible* if it is not visible—formally, the predicate $\mathsf{inv}^{Sc,obs}_{ml}\, m$ is defined to mean $m \in \mathsf{Avail}^{Sc,obs}\, ml \setminus \mathsf{visAvail}^{Sc,obs}\, ml$.

A scheduler shall be declared noninterfering if the effect of removing invisible threads is the same as that of hiding them. This property can be formulated in a manner rather faithful to the style of G&M [5] (recalled in the introduction). Our "users" of the system managed by *sch* are the threads (thread IDs), and at each history point there are two groups of users, visible and invisible, and thus we require that the *observations* of the visible threads do not depend on the *actions* of the invisible ones. Clearly, the actions should be steps taken by the threads. Moreover, we choose the observation of a visible user n at history ml to be the probability that n will be scheduled first among all the visible threads, i.e., the "exit probability" of n after zero or more invisible steps, $\mathsf{P}^{Sc,sch}_{ml}\,(\mathsf{Takes}\ \mathsf{inv}^{Sc,obs}\ \mathsf{Until}\ \mathsf{Takes}\ n)$. Note that here, unlike in [5], current users may disappear (by termination) and new users may appear (by spawning), and therefore $\mathsf{inv}^{Sc,obs}$ is not a fixed set, but a set evolving over time; this is properly handled by the history-dependent interpretation of temporal formulas.

Having the observations and the actions in place, it is time to zoom in the definition of noninterference from [5] in more technical detail: For all users n of the second group (here, the visible threads), the observation of n based on the history (where "history" means, in [5] as well as here, "the sequence of actions the users have taken in the past") is required to be the same as the observation of n on the restriction of the history by removing all actions of users from the first group (here, the invisible threads). In order to formally perform this removal, i.e., filter out the (Sc,obs)-invisible actions from histories, we first define recursively $\mathsf{visHist}^{Sc,obs}\, ml$, the visible restriction of a history ml:

$$\mathsf{visHist}^{Sc,obs}[] = [] \qquad \mathsf{visHist}^{Sc,obs}(ml \,\#\, m) = \begin{cases} (\mathsf{visHist}^{Sc,obs}\, ml) \,\#\, m, & \text{if } m \in \mathsf{visAvail}^{Sc,obs}\, ml, \\ \mathsf{visHist}^{Sc,obs}\, ml, & \text{otherwise.} \end{cases}$$

Moreover, we collect the available visible threads throughout history ml in the set $\mathsf{visHavail}^{Sc,obs}\, ml$ also defined recursively:

$$\mathsf{visHavail}^{Sc,obs}\,[] = [\{\varepsilon\}]$$

$$\mathsf{visHavail}^{Sc,obs}\,(ml \,\#\, m) = \begin{cases} (\mathsf{visHavail}^{Sc,obs}\, ml) \,\#\, (\mathsf{visAvail}^{Sc,obs}\, ml), & \text{if } m \in \mathsf{visAvail}^{Sc,obs}\, ml, \\ \mathsf{visHavail}^{Sc,obs}\, ml, & \text{otherwise.} \end{cases}$$

The scheduler *sch* is called *noninterfering* if the following holds for all OA-scenarios (Sc,obs), all $ml \in Sc$, and all $n \in \mathsf{visAvail}^{Sc,obs}\, ml$:

$$\mathsf{P}^{Sc,sch}_{ml}\,(\mathsf{Takes}\ \mathsf{inv}^{obs,Sc}\ \mathsf{Until}\ \mathsf{Takes}\ n) = sch_{ml',Ml'}\, n,$$

where $(ml', Ml') = (\mathsf{visHist}^{Sc,obs}\, ml, \mathsf{visHavail}^{Sc,obs}\, ml)$.

In the above equality, the lefthand side expresses the observation made by n at history ml, and the righthand side the observation that n would make if any trace of invisible threads were removed (from both the history and the available threads). Since our notion of observation effectively hides invisible threads (in the style of τ-actions form process algebra), the meaning of the above equality can be summarized as

Removal = Hiding (of invisible threads)

Note that the notion of scheduler noninterference is independent of the concrete notion of command at the expense of quantifying universally over all scenarios.

An important question is whether a reasonable class of schedulers are noninterfering. Roughly speaking, any scheduler that is "politically correct", treating its threads uniformly, is noninterfering.

Proposition 1. The uniform and round-robin schedulers from §2.3 are noninterfering.

Proof idea. We fix (Sc, obs) and $ml \in Sc$. We need to show the equality of two functions defined on $n \in \mathsf{visAvail}\ ml$, say $F = G$, where $F\ n = \mathsf{P}_{ml}^{Sc,sch}\ (\mathsf{Takes\ inv}^{obs,Sc}\ \mathsf{Until\ Takes}\ n)$ and $G\ n = sch_{ml',Ml'}\ n$.

For usch, noninterference follows immediately from its symmetry, since both F and G are constant on visAvail ml. For rsch^j, let m be the last thread in ml and $k = \$(ml)$. If m is visible and $k < j$, then both $F\ n$ and $G\ n$ are either 1, if $n = m$, or 0, otherwise. If m is invisible or $k \geq j$, then both $F\ n$ and $G\ n$ are either 1, if n is the next visible thread in the queue, or 0, otherwise. □

Several other noninterfering schedulers are presented in [10, §A]. It is also instructive to see an *interfering* one: Consider a modification of the round robin that increments the quota at each shift to a new thread. Then consider the history $ml = [n_1, m, m, n_2, n_2]$ with n_1, n_2 visible and m invisible. Since n_2 still has one step in its quota, $F\ n_2 = 1$. On the other hand, $ml' = [n_1, n_2, n_2]$, meaning that, at ml', n_2 yields to n_1, hence $G\ n_2 = 0$.

4.2 Possibilistic Noninterference

To discuss noninterference of commands, we fix an attacker-observation function aobs : **state** → **odom**. A typical choice of aobs [23] assumes the state consists of values stored in variables classified as high-security or low-security and defines aobs to return the low-variable part of the state (see [10, §D]). We define possibilistic noninterference by a form of bisimilarity up to invisibility.

Invisibility of a command is defined as "never change the observation on the state", technically, coinductively as the weakest predicate invis satisfying the following property: for all $c, s, \gamma, c_1, \ldots, c_l, s'$ such that invis c and $c \xrightarrow{s} (\gamma, [c_1, \ldots, c_l], s')$, we have that: **(1)** aobs s' = aobs s, **(2)** invis c_i for all $i \in \{1, \ldots, k\}$; **(3)** $\gamma \in$ **cmd** implies invis γ.

Possibilistic bisimilarity of two commands is now defined coinductively as the weakest relation $\approx$ satisfying the following property: for all c, d, if $c \approx d$, then either invis c and invis d or, for all $s, t, \gamma, c_1, \ldots, c_l, s', \delta, d_1, \ldots, d_k, t'$ such that aobs s = aobs t, $c \xrightarrow{s} (\gamma, [c_1, \ldots, c_l], s')$ and $d \xrightarrow{t} (\delta, [d_1, \ldots, d_k], t')$, we have that: **(1)** aobs s' = aobs t'; **(2)** $l = k$ and $c_i \approx d_i$ for all $i \in \{1, \ldots, l\}$; **(3)** if $\gamma = \bot$, then either $\delta = \bot$ or invis δ; **(4)** if $\delta = \bot$, then either $\gamma = \bot$ or invis γ; **(5)** if $\gamma, \delta \in$ **cmd**, then $\gamma \approx \delta$.

A command c is called *possibilistically noninterfering* if $c \approx c$. Thus, possibilistic noninterference of c means that alternative executions starting in states indistinguishable by the attacker proceed in a synchronized manner for as long as one of them does not reach an invisible status, moment at which the other is required to also reach such a status; moreover, termination should be matched by either termination or invisibility.

The componentwise extension of this notion to thread pools coincides with the flexible scheduler-independent security introduced by Mantel and Sudbrock in [9]. As argued in [9], this notion is both compositional and flexible enough to allow the execution time of programs to depend on secrets. However, it does share the common limitation of PER approaches [15] aimed at scheduler independence: its rather strong lock-step synchronization nature (albeit only on visible executions).

Note that the example from the introduction does not satisfy possibilistic noninterference since, depending on the initial value of h, one alternative execution may enable the visible action $l := 2$ earlier than another alternative execution. And indeed, our intention with possibilistic noninterference is to guard (in the presence of noninterfering schedulers) against probabilistic attacks of the kind allowed by this program—this will be our main result, Th. 2.

4.3 Probabilistic Noninterference

We define probabilistic noninterference following the weak bisimulation approach taken by Smith [18], using an adaptation of a corresponding notion from probabilistic process algebra due to Baier and Hermanns [2]: Roughly, a command shall be deemed probabilistically noninterfering if any two executions of it starting in states that differ only on secret information traverse the same sequence of attacker observations with the same probabilities. As argued in [18, p.11], this notion is suitable for protecting against internal leaks, but not external leaks such as timing.

In our formalism, we can define everything in terms of scenarios and their scheduler-induced Markov chains. Indeed, the function $\mathsf{config}_{c,s}$ introduced in §3.2 "observes", at each execution history ml, the whole thread pool configuration. The attacker's observations on execution histories shall be much more restricted: only the state can be observed, and only through aobs. Namely, assuming $\mathsf{config}_{c,s}\, ml = (ml, Ml, thr, t)$, we define $\mathsf{obs}_{c,s}\, ml = \mathsf{aobs}\, t$. Thus, the OA-scenario $(Sc_{c,s}, \mathsf{obs}_{c,s})$ is a description of the executions of command c starting in state s, as observed by the attacker.

Given $H, H' \subseteq Sc$ and $ml \in Sc_{c,s}$, we define $ml \Rightarrow_H H'$ to be the set of all traces that go through elements of H only and eventually reach an element of H', namely,

$$\{mt \in \mathsf{Trace}_{c,s}.\ \exists nl'.\ [] \neq nl' \preceq mt \ \wedge\ (\forall nl \prec nl'.\ ml \# nl \in H) \ \wedge\ ml \# nl' \in H'\},$$

where $\prec$ and $\preceq$ denote the strict and nonstrict prefix orderings on finite or infinite sequences. Note that $ml \Rightarrow_H H'$ is empty unless $ml \in H$.

Given an equivalence relation E, Cls_E denotes its set of equivalence classes, which we simply call *E-classes*. Let (Sc, obs) be an OA-scenario. A relation $E : Sc_{c,s} \to Sc_{c,s} \to$ **bool** is called a *sch-probabilistic bisimulation* for (c, s) if the following hold:

(I1) E is an equivalence relation on $Sc_{c,s}$ with countable set of equivalence classes.

(I2) For $ml, nl \in Sc_{c,s}$, $E\ ml\ nl$ implies $\mathsf{obs}_{c,s}\ ml = \mathsf{obs}_{c,s}\ nl$.
(I3) For distinct E-classes H, H' and $ml, nl \in H$, $\mathsf{P}^{sch,c,s}_{ml}(ml \Rightarrow_H H') = \mathsf{P}^{sch,c,s}_{nl}(nl \Rightarrow_H H')$.

Thanks to condition (I3), we can define, for any two distinct E-classes H and H', $\mathsf{P}^{sch,c,s}(H \Rightarrow H')$, *the probability of moving from H directly to H′ (without visiting any other E-class)*, to be $\mathsf{P}^{sch,c,s}_{ml}(ml \Rightarrow_H H')$ for some (any) $ml \in H$. Thus, a probabilistic bisimulation E provides a class partition of the scenario (I1) so that elements of the same class are indistinguishable both w.r.t. observations (I2) and probabilistic behavior (I3). By (I2), an attacker is only able to observe the sequence of E-classes induced by an execution; by (I3), this sequence is statistically the same (modulo repetition) regardless of the concrete E-class representatives.

We also define a binary version of this indistinguishability relation. (c,s) and (c',s') are called *sch-probabilistically bisimilar* if there exist E, E', F such that:

(1) E and E' are *sch*-probabilistic bisimulations for (c,s) and (c',s'), respectively.
(2) $F : \mathsf{Cls}_E \to \mathsf{Cls}_{E'}$ is a bijection such that
  (a) $\mathsf{obs}_{c,s}\ ml = \mathsf{obs}_{c',s'}\ ml'$ for all ml, ml', H with $ml \in H \in \mathsf{Cls}_E$ and $ml' \in F\ H$,
  (b) $\mathsf{P}^{sch,c,s}(H \Rightarrow H_1) = \mathsf{P}^{sch,c',s'}(F\ H \Rightarrow F\ H_1)$ for all $H, H_1 \in \mathsf{Cls}_E$,
  (c) $F\ H_0 = H'_0$, where H_0 and H'_0 are the equivalence classes of $[]$ in Cls_E and $\mathsf{Cls}_{E'}$.

Finally, c is called *sch-probabilistically noninterfering* if (c,s) and (c,s') are *sch*-probabilistically bisimilar for all s, s' such that $\mathsf{aobs}\ s = \mathsf{aobs}\ s'$.

4.4 Noninterference Criterion

We can now state our main result connecting three concepts that were defined mutually independently: possibilistic and probabilistic noninterference of commands and noninterference of schedulers.

Theorem 2. If *sch* is noninterfering and c is possibilistically noninterfering, then c is *sch*-probabilistically noninterfering.

Proof idea. The key of the proof consists of the definition, for any OA-scenario, of its visible sub-OA-scenario obtained from removing, at each history, the currently invisible threads. The latter can be proven probablistically bisimilar to the original OA-scenario, the bisimilarity step being handled using the noninterference of *sch*. Moreover, from the noninterference of c, it follows that $Sc_{c,s}$ and $Sc_{c,s'}$ have the same visible sub-OA-scenario Sc', which makes them probabilistically bisimilar. (See [10, §E].) □

Next we discuss the security requirements and guarantees of this theorem.

Requirement 1 (R1): Scheduler noninterference. This is a background condition that needs to be verified for the scheduler once and for all. Its verification involves quantitative computation with probabilities. However, it is a natural condition expressing a certain symmetry of the scheduler, and its verification tends to be easy for the examples considered in §2.3 (as well as for other examples described in [10, §A]).

Requirement 2 (R2): Possibilistic noninterference. Unlike R2, this condition needs to be verified for each individual program. Fortunately, this style of PER properties is amenable for compositional verification [8, 15, 17, 18]. In particular, the type systems

from [3,4,9,18], as well as the harsher ones from [17,20], are static criteria on multi-threaded programs enforcing this property.

Guarantee (G): Probabilistic noninterference. This appears to be the strongest security guarantee of a probabilistic system provided we ignore timing channels [18]: An attacker making observations of the low part of the memory while the program by multiple running cannot infer any secret, not even by statistically from multiple runs.

In order to further comprehend (G), let us have a look at a consequence in terms of end-to-end security. Given c, s and $S \in \mathbf{odom}$, we define $\mathsf{endUpIn}_{c,s}\, S \subseteq \mathsf{Trace}_{[]}^{sch,c,s}$ as the set of traces that eventually "end up in S", i.e., that eventually reach a point where the attacker observation becomes S and stays constantly S—in LTL, this set is described by the formula $\mathsf{Ev}\,(\mathsf{Alw}\,\mathsf{obs}_{c,s}^{-1})$. Note that the traces in $\mathsf{endUpIn}_{c,s}\, S$ need not be terminating.

Proposition 3. If c is *sch*-probabilistically noninterfering, then, for all c, s, s', S, $\mathsf{aobs}\ s = \mathsf{aobs}\ s'$ implies $\mathsf{P}_{[]}^{sch,c,s}\,(\mathsf{endUpIn}_{c,s}\, S) = \mathsf{P}_{[]}^{sch,c,s'}\,(\mathsf{endUpIn}_{c,s'}\, S)$.

The guarantee of Prop. 3 is that executions starting in indistinguishable states stabilize in any given attacker-indistinguishable class S with the same probabilities. Note that termination implies stabilization (but not vice versa), so in particular Prop. 3 says that if the two executions terminate, then the resulted states have the same probability distribution w.r.t. what the attacker can see.

Example. We assume that programs are specified in a simple while language with thread-spawning facilities, states are assignments of values to variables, variables are classified into low and high, and the attacker observation is the low part of the state. Consider the following multi-threaded program adapted from [9, §5.2]:

$$\mathsf{while\ True\ do}\ \{l_1 := \mathit{inp}_1\ ;\ l_2 := \mathit{inp}_2\ ;\ \mathsf{spawn}\ T\ ;\ \mathsf{spawn}\ T_1\ ;\ \mathsf{spawn}\ T_2\}$$

where T is $h := \mathsf{addH}(l_1, h)$, T_1 is $l := \mathsf{addL}(l_1, l)$, and T_2 is $l := \mathsf{addL}(l_2, l)$.

The program repeatedly performs the following actions: It receives two public values (through input channels modeled here as low variables inp_1 and inp_2 assumed to be volatile) and stores them in the low variables l_1 and l_2. Then it spawns three threads, T, T_1, T_2. T applies the non-atomic operation addH for updating a private database h with l_1, whose timing depends on the value of h. T_1 and T_2 apply the atomic operation addL for updating a public database l with l_1 and l_2, respectively.

This is an intuitively secure program w.r.t. time-insensitive attacks: regardless of the values of the low variables, the execution of the main thread takes the same path, repetitively spawning copies of T_1, T_2 (that assign low to low) and T (that assigns low to high); the execution of T does depend on h, but this is harmless, since T does not affect the low part of the state or the behavior of the other threads. The program is automatically checked to be possibilistically noninterfering by existing type systems [9, 18] (see also [10, §D]). Our Th. 2 ensures that it is also noninterfering if run under any noninterfering scheduler, in particular, the uniform and round robin ones.

This was a simple example of a kind widely encountered in web computing and operating systems: nonterminating multi-threaded programs providing a form of service. However, it is not proved noninterfering by previous scheduler-independent criteria. In particular, it does not satisfy strong security [17] (since the running time T may depend

on secrets) or observational determinism [24] (since T_1 and T_2 are in a data race). Due to nontermination, it also falls outside the scope of the criterion from [9].

5 Conclusions and Related Work

In this paper, we proposed a novel notion of scheduler noninterference, which was proved to behave securely w.r.t. refinement of nondeterminsim: possibilistic noninterference of the multi-threaded program implies probabilistic noninterference when run under the given scheduler. We have *not* introduced novel notions of possibilistic or probabilistic noninterference, but used (minor adaptations of) existing ones [9, 18]. Consequently, we can employ existing syntactic methods for verifying that programs satisfy the hypothesis of our main result, Th. 2.

Mantel and Sudbrock [9] define flexible scheduler-independent (FSI) security, which we use as our possibilistic noninterference. They also introduce the class of *robust* schedulers and they prove an end-to-end security property, in the style of our Prop. 3, but conditioned by termination of the program. As already discussed, a major improvement of our Th. 2 is freeness from the termination assumption which is both hard to check and often not true. Even ignoring termination, the security guarantee of Th. 2 is significantly stronger than that of [9], as it takes into account the whole sequence of attacker observations throughout execution, and not only at the end of it. Another difference between our setting and [9] is the considered class of schedulers. Like our noninterfering schedulers, the robust schedulers were shown to include the round robin and uniform ones. However, robust schedulers are introduced via a probabilistic simulation relation involving both the scheduler and FSI-secure thread pools. Our noninterference condition for schedulers has a more natural justification in terms of G&M noninterference and is stated in isolation from the concrete operational semantics of threads (although it does employ thread ID interleavings); arguably, it is also easier to check. On the other hand, the notion of scheduler from [9] allows the flexibility of an arbitrary scheduler state—we could not have employed the history-based G&M noninterference had we worked with such general schedulers.

Smith [18] defines probabilistic noninterference via weak probabilistic bisimulation and provides a type system criterion for it, assuming the uniform scheduler. Since the guarantee of Th. 2 is precisely Smith's probabilistic noninterference, our result is in effect a generalization of his results to a wide class of schedulers.

Sabelfeld and Sands [17] introduce *strong security* for thread pools (a PER notion requiring complete lock-step synchronization of alternative executions) and prove security w.r.t. *all* schedulers; moreover, Sabelfeld [15] proves that strong security cannot be weakened if we are after a *compositional* notion covering the whole class of schedulers. Zdancewic and Myers [24] take a whole different approach to scheduler independence, focusing on concurrent programs that are a priori safe under refinement attacks, in that the attacker's sequence of observations is the same in any execution (observational determinism). This is achieved practically by a data race freedom analysis in conjunction with a type system. Boudol and Castellani [4] describe yet another approach, based on an operational semantics for the scheduler, run in parallel with a thread pool that it controls. They do not cover probabilistic schedulers or dynamic thread creation. Finally, Russo and Sabelfeld [13] achieve scheduler independence by allowing

the threads to explicitly change their security levels and the scheduler to discriminate between threads according to their levels. [13, §2] and [9, §6] survey more work on scheduler-independent security. Unlike here, previous work [9, 17] allows schedulers to depend on the low part of the state. This is also possible in our framework and is pursued in [10, §A], but here it has been omitted as it brings no further insight into our method.

This paper was concerned with lifting possibilistic noninterference to probabilistic noninterference. Somewhat complementary, our previous work [11] studies and classifies various notions of possibilistic noninterference and their compositionality w.r.t. language constructs.

Acknowledgment. We thank Jasmin Blanchette and the referees for useful comments and suggestions. This work was supported by the DFG project Ni 491/13–2, part of the DFG priority program Reliably Secure Software Systems (RS^3).

References

1. Agat, J.: Transforming out timing leaks. In: POPL, pp. 40–53 (2000)
2. Baier, C., Hermanns, H.: Weak bisimulation for fully probabilistic processes. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 119–130. Springer, Heidelberg (1997)
3. Boudol, G., Castellani, I.: Noninterference for concurrent programs. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 382–395. Springer, Heidelberg (2001)
4. Boudol, G., Castellani, I.: Noninterference for concurrent programs and thread systems. Theoretical Computer Science 281(1-2), 109–130 (2002)
5. Goguen, J.A., Meseguer, J.: Security policies and security models. In: IEEE Symposium on Security and Privacy, pp. 11–20 (1982)
6. Kemeny, J.G., Snell, J.L., Knapp, A.W.: Denumerable Markov chains, 2nd edn. Springer (1976)
7. Mantel, H., Sabelfeld, A.: A generic approach to the security of multi-threaded programs. In: CSFW, pp. 200–214 (2001)
8. Mantel, H., Sands, D., Sudbrock, H.: Assumptions and guarantees for compositional noninterference. In: CSF 2001, pp. 218–232 (2011)
9. Mantel, H., Sudbrock, H.: Flexible scheduler-independent security. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) ESORICS 2010. LNCS, vol. 6345, pp. 116–133. Springer, Heidelberg (2010)
10. Popescu, A., Hölzl, J., Nipkow, T.: Noninterfering schedulers—when posibilistic noninterference implies probabilistic noninterference. Technical report. Technische Universität München (2013), `http://mediatum.ub.tum.de/?id=1159789`
11. Popescu, A., Hölzl, J., Nipkow, T.: Proving concurrent noninterference. In: Hawblitzel, C., Miller, D. (eds.) CPP 2012. LNCS, vol. 7679, pp. 109–125. Springer, Heidelberg (2012)
12. Russo, A., Hughes, J., Naumann, D., Sabelfeld, A.: Closing internal timing channels by transformation. In: Okada, M., Satoh, I. (eds.) ASIAN 2006. LNCS, vol. 4435, pp. 120–135. Springer, Heidelberg (2008)
13. Russo, A., Sabelfeld, A.: Securing interaction between threads and the scheduler. In: IEEE Computer Security Foundations Workshop, pp. 177–189 (2006)

14. Russo, A., Sabelfeld, A.: Security for multithreaded programs under cooperative scheduling. In: Virbitskaite, I., Voronkov, A. (eds.) PSI 2006. LNCS, vol. 4378, pp. 474–480. Springer, Heidelberg (2007)
15. Sabelfeld, A.: Confidentiality for multithreaded programs via bisimulation. In: Broy, M., Zamulin, A.V. (eds.) PSI 2003. LNCS, vol. 2890, pp. 260–274. Springer, Heidelberg (2004)
16. Sabelfeld, A., Myers, A.C.: Language-based information-flow security. IEEE Journal on Selected Areas in Communications 21(1), 5–19 (2003)
17. Sabelfeld, A., Sands, D.: Probabilistic noninterference for multi-threaded programs. In: IEEE Computer Security Foundations Workshop, pp. 200–214 (1999)
18. Smith, G.: Probabilistic noninterference through weak probabilistic bisimulation. In: IEEE Computer Security Foundations Workshop, pp. 3–13 (2003)
19. Smith, G.: Improved typings for probabilistic noninterference in a multi-threaded language. Journal of Computer Security 14(6), 591–623 (2006)
20. Smith, G., Volpano, D.: Secure information flow in a multi-threaded imperative language. In: ACM Symposium on Principles of Programming Languages, pp. 355–364 (1998)
21. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification (preliminary report). In: LICS, pp. 332–344 (1986)
22. Volpano, D., Smith, G.: Probabilistic noninterference in a concurrent language. Journal of Computer Security 7(2-3), 231–253 (1999)
23. Volpano, D., Smith, G., Irvine, C.: A sound type system for secure flow analysis. Journal of Computer Security 4(2-3), 167–187 (1996)
24. Zdancewic, S., Myers, A.C.: Observational determinism for concurrent program security. In: IEEE Computer Security Foundations Workshop, pp. 29–43 (2003)

Simulations and Bisimulations for Coalgebraic Modal Logics

Daniel Gorín and Lutz Schröder

Department of Computer Science, Universität Erlangen-Nürnberg

Abstract. Simulations serve as a proof tool to compare the behaviour of reactive systems. We define a notion of Λ-simulation for coalgebraic modal logics, parametric in the choice of a set Λ of monotone predicate liftings for a functor T. That is, we obtain a generic notion of simulation that can be flexibly instantiated to a large variety of systems and logics, in particular in settings that semantically go beyond the classical relational setup, such as probabilistic, game-based, or neighbourhood-based systems. We show that this notion is adequate in several ways: i) Λ-simulations preserve truth of positive formulas, ii) for Λ a separating set of monotone predicate liftings, the associated notion of Λ-bisimulation corresponds to T-behavioural equivalence (moreover, this correspondence extends to the respective finite-lookahead counterparts), and iii) Λ-bisimulations remain sound when taken up to difunctional closure. In essence, we arrive at a modular notion of equivalence that, when used with a separating set of monotone predicate liftings, coincides with T-behavioural equivalence regardless of whether T preserves weak pullbacks. That is, for finitary set-based coalgebras, Λ-bisimulation works under strictly more general assumptions than T-bisimulation in the sense of Aczel and Mendler.

1 Introduction

Coalgebra provides an abstract framework for the unified study of a wide variety of reactive systems beyond the classical relational setup, including probabilistic, weighted, neighbourhood-based, game-based, and preferential systems. As the basic notion of equivalence in coalgebra, T-behavioural equivalence has emerged, which declares two states to be equivalent if they are identified by some pair of coalgebra morphisms; in case the type functor T admits a final coalgebra, T-behavioural equivalence is just identification in the final T-coalgebra. As a proof principle, however, T-behavioural equivalence is comparatively unwieldy, thus motivating the search for bisimulation-type proof principles whereby two states can be shown to be behaviourally equivalent by exhibiting a *bisimulation relation* between them. The advantage of such approaches is that bisimulation relations may be comparatively small, making equivalence proofs by bisimulation more manageable than direct proofs of behavioural equivalence.

The downside is that while behavioural equivalence is a canonical notion that works for any type of coalgebras, it is rather less clear what a bisimulation is in general. In case the type functor preserves weak pullbacks, the standard notion of T*-bisimulation* gives a satisfactory answer: it can be uniformly defined for any T, it is always sound for T-behavioural equivalence, given that T preserves weak pullbacks it is complete for T-behavioural equivalence, and it coincides with standard notions in the main examples.

R. Heckel and S. Milius (Eds.): CALCO 2013, LNCS 8089, pp. 253–266, 2013.

For functors that fail to preserve weak pullbacks, however, the search for a good generic notion of bisimilarity remains largely open.

Here, we present a modally inspired notion of bisimulation that partly solves these problems; specifically it does so for functors that admit a separating set of *monotone* predicate liftings. This includes, but is not limited to, all finitary functors that preserve weak pullbacks [5], thus properly extending the family of functors covered by T-bisimulations in the finitary case (a separating example being the finitary part of the monotone neighbourhood functor). Our notion of Λ-*bisimilarity* depends on distinguishing a modal signature Λ that we assume to consist of monotone operators. Key features of Λ-bisimilarity are

- It is related to a corresponding notion of Λ-*simulation*, which bears a clear relation to modal logic: all positive modal formulas over Λ are preserved by Λ-simulations.
- If Λ is separating, then Λ-bisimulation is sound and complete for behavioural equivalence.
- We have a finite-lookahead version of Λ-bisimilarity. This Λ-n-bisimilarity is sound and complete for the standard notion of n-behavioural equivalence defined via the terminal sequence.
- Λ-bisimulation allows bisimulation proofs up to difunctionality (i.e. closure under zig-zags).
- Λ-bisimulations are more general than T-bisimulations, i.e. every T-bisimulation is a Λ-bisimulation (the converse holds for difunctional relations when T preserves weak pullbacks and Λ is separating).

2 Preliminaries

The framework of *coalgebraic modal logic* [9] covers a broad range of modalities including modal operators with non-relational semantics, such as probabilistic and game-theoretic phenomena as well as neighbourhood semantics and non-material conditionals [13]. This framework is parametric in syntax and semantics. The syntax is given by a *similarity type* Λ, i.e. a set of *modal operators* with finite arities ≥ 0 (hence possibly including propositional atoms). To simplify notation, we will pretend that all operators are unary.

Definition 1. The set $L(\Lambda)$ of Λ-*formulas* is given by the grammar:

$$\phi, \psi ::= \top \mid \neg\phi \mid \phi \wedge \psi \mid \heartsuit\phi \qquad (\heartsuit \in \Lambda).$$

We use the standard derived Boolean operators $\vee$, $\rightarrow$, etc. We use $rank(\phi)$ to denote the maximum number of nested occurrences of $\heartsuit \in \Lambda$ in ϕ.

The semantics is parametrized by associating a Λ-structure $\langle T, \{[\![\heartsuit_\lambda]\!]\}_{\lambda\in\Lambda}\rangle$ to a similarity type Λ. Here T is an endofunctor on the category Set and each $[\![\heartsuit_\lambda]\!]$ is a *predicate lifting*, that is, a natural transformation $[\![\heartsuit]\!] : \mathcal{Q} \dot{\rightarrow} \mathcal{Q} \circ T^{op}$, where $\mathcal{Q}$ is the contravariant powerset functor $\mathsf{Set}^{op} \rightarrow \mathsf{Set}$ (that is, $\mathcal{Q}X = 2^X$ for every set X, and given $f : X \rightarrow Y$, $\mathcal{Q}f : 2^X \rightarrow 2^Y$ is given by $\mathcal{Q}f(A) = f^{-1}[A]$). For the extension of predicate liftings to the higher-arity case see [14].

Assumption 2. We can assume w.l.o.g. that T preserves injective maps [1]. For convenience of notation, we will in fact sometimes assume that subset inclusions $X \hookrightarrow Y$ are mapped to subset inclusions $TX \hookrightarrow TY$. Moreover, we assume w.l.o.g. that T is non-trivial, i.e. $TX = \emptyset \implies X = \emptyset$ (otherwise, $TX = \emptyset$ for all X).

We typically identify a similarity type Λ and its associated Λ-structure, and refer to both as Λ. Unless otherwise stated, T stands for the underlying functor of the given Λ-structure.

For a given choice of Λ, a model for $L(\Lambda)$ is just a T-*coalgebra* $\langle X, \xi \rangle$, i.e. a non-empty set X (the set of *states*) and a *transition function* $\xi : X \to TX$. Given $x \in X$, the truth value of $L(\Lambda)$-formulas is defined as:

$$x \models_\xi \top \qquad \text{always} \tag{1}$$

$$x \models_\xi \neg\phi \iff x \not\models_\xi \phi \tag{2}$$

$$x \models_\xi \phi \wedge \psi \iff x \models_\xi \phi \text{ and } x \models_\xi \psi \tag{3}$$

$$x \models_\xi \heartsuit\phi \iff \xi(x) \models \heartsuit[\![\phi]\!]_\xi \tag{4}$$

where $[\![\phi]\!]_\xi$, the extension of ϕ in ξ, is given by $[\![\phi]\!]_\xi = \{x \in X \mid x \models_\xi \phi\}$, and for $t \in TX$ and $A \subseteq X$, $t \models \heartsuit A$ is a more suggestive notation for $t \in [\![\heartsuit]\!]_X A$. When clear from the context, we shall write simply $x \models \phi$ and $[\![\phi]\!]$.

Example 3. Coalgebras for the (covariant) finite powerset functor $\mathcal{P}_\omega$ are finitely branching directed graphs. For a similarity type $\Lambda = \{\Box, \Diamond\}$ consider the associated predicate liftings:

$$[\![\Box]\!]_X(A) := \{B \mid B \subseteq A\} \tag{5}$$

$$[\![\Diamond]\!]_X(A) := \{B \mid B \cap A \neq \emptyset\} \tag{6}$$

They correspond to the classical modal operators of relational modal logics, so the logic we get in this case is essentially the mono-modal version of Hennessy-Milner logic [4]. To obtain the basic modal logic K one needs to enrich the coalgebra structure with an interpretation for propositions. So let V be a set of proposition symbols and let C_V be the constant functor that maps every set X to 2^V. For each $p \in V$, the (nullary) predicate lifting $[\![p]\!]_X := \{\pi \in 2^V \mid p \in \pi\}$ describes structures satisfying p. The Kripke functor K is then defined as $KX := C_V \times \mathcal{P}X$ and the similarity type $\Lambda = V \cup \{\Diamond, \Box\}$ is interpreted using the corresponding predicate liftings on the appropriate projections.

Example 4. The language of *graded modal logic* corresponds to the similarity type $\Lambda = \{\Diamond_k \mid k \in \mathbb{N}\}$ and is interpreted over the infinite multiset functor $\mathcal{B}_\infty$, i.e., $\mathcal{B}_\infty X = X \to \mathbb{N} \cup \{\infty\}$. Coalgebras for $\mathcal{B}_\infty$ are *multigraphs*, i.e. directed graphs whose edges are annotated with (potentially infinite) non-negative integer multiplicities. Interpretation of the modal operators is by way of the following family of predicate liftings, for each $k \in \mathbb{N}$:

$$[\![\Diamond_k]\!]_X(A) := \{b \in \mathcal{B}_\infty X \mid b(A) > k\} \tag{7}$$

where by $b(A)$ we denote $\sum_{x \in A} b(x)$, i.e. we use $b \in \mathcal{B}_\infty X$ like a measure on X.

Example 5. Probabilistic modal logics are obtained when one takes the functor $\mathcal{D}$ that maps X to the set of discrete probability distributions over X. For the language $\Lambda_M = \{M_p \mid p \in [0,1] \cap \mathbb{Q}\}$, with M_p informally read as "with probability more than p", the corresponding predicate liftings are defined analogously as for graded modal logics. One can instead take $\Lambda_P = \{L_p \mid p \in [0,1] \cap \mathbb{Q}\}$, with L_p read as "with probability at least p", and interpreted using

$$[\![L_p]\!]_X(A) := \{\mu \in \mathcal{D}X \mid \mu(A) \geq p\}.$$

Example 6. As a final example, consider the subfunctor $\mathcal{M}$ of the neighbourhood functor $\mathcal{Q} \circ \mathcal{Q}$ given by $\mathcal{M}X = \{S \in \mathcal{Q}\mathcal{Q}X \mid S \text{ is upwards closed}\}$. Over this functor one can obtain the monotone neighborhood semantics of modal logic with $\Lambda = \{\Box\}$ using the predicate lifting $[\![\Box]\!]_X(A) := \{S \in \mathcal{M}X \mid A \in S\}$.

A modal operator $\heartsuit$ is called *monotone* if it satisfies the condition

$$A \subseteq B \subseteq X \text{ implies } [\![\heartsuit]\!]_X A \subseteq [\![\heartsuit]\!]_X B.$$

While all the examples above are monotone, it is worth stressing that the framework of coalgebraic modal logics can indeed accommodate non-monotone logics. We will however focus on the monotone case.

Assumption 7. In the following, we assume all modal operators to be monotone.

For a given endofunctor T, the choice of both the similarity type Λ and the associated Λ-structure over T may vary (although the number of choices is formally limited [14]), and each choice yields a potentially different logic. When the choice of predicates of liftings in Λ is rich enough as to uniquely describe every element in TX, we call such Λ *separating* [10]:

Definition 8. We say that Λ is *separating* if $t \in TX$ is uniquely determined by the set $\{(\heartsuit, A) \in \Lambda \times \mathcal{P}X \mid t \models \heartsuit A\}$.

It is not hard to see that, for example, $V \cup \{\Box\}$ as well as $V \cup \{\Diamond\}$ are separating over the Kripke functor K of Example 3. The reader is referred to [14] for characterizations of functors that admit separating sets of predicate liftings.

Definition 9. Given T-coalgebras C and D, we say that states x in C and y in D are *behaviourally equivalent*, and write $(C, x) \approx (D, y)$, or shortly $x \approx y$, whenever there exists a T-coalgebra E and coalgebra morphisms $f : C \to E$ and $g : D \to E$ such that $f(x) = g(y)$.

Simulations like the ones we will present in Section 3 occur frequently when dealing with logics that do not contain a Boolean basis; typically, negation is absent or only allowed on restricted positions (e.g., in front of atoms). The notion of *positive* formula is a generalization of this idea.

Definition 10. The language $L^+(\Lambda)$ of *positive Λ-formulas* is given by:

$$\phi, \psi ::= \top \mid \bot \mid \phi \wedge \psi \mid \phi \vee \psi \mid \heartsuit\phi \qquad (\heartsuit \in \Lambda).$$

We can regard $L^+(\Lambda)$ as a syntactic fragment of $L(\Lambda)$ where $\vee$ is now taken as primitive. The Boolean connectives of $L^+(\Lambda)$ allow expressing all the monotone Boolean functions, but notice that Λ may contain dual operators (e.g., $\Lambda = \{\Box, \Diamond\}$) — in fact if Λ is closed under dual operators then $L^+(\Lambda)$ is as expressive as $L(\Lambda)$. In general, of course, $L^+(\Lambda)$ is a proper fragment of $L(\Lambda)$.

3 Coalgebraic Simulation

We now proceed to introduce our notion of modal simulation. We use standard notation for relations; in particular, given a binary relation $S \subseteq X \times Y$ and $A \subseteq X$, we denote by $S[A]$ the relational image $S[A] = \{y \mid \exists x \in A.\, xSy\}$.

Definition 11 (Λ-Simulation, Λ-Homomorphism). Let $C = (X, \xi)$ and $D = (Y, \zeta)$ be T-coalgebras. A *Λ-simulation* $S : C \to D$ *(of D by C)* is a relation $S \subseteq X \times Y$ such that whenever xSy then for all $\heartsuit \in \Lambda$ and all $A \subseteq X$

$$\xi(x) \models \heartsuit A \text{ implies } \zeta(y) \models \heartsuit S[A].$$

A function $f : X \to Y$ is a *Λ-homomorphism* if its graph is a Λ-simulation.

Lemma 12. *Λ-simulations are stable under unions and relational composition. Moreover, equality is always a Λ-simulation.*

Definition 13 (Λ-ordering). The *Λ-preorder* $\leq_\Lambda$ on TX is defined by

$$s \leq_\Lambda t \iff \forall \heartsuit \in \Lambda, A \subseteq X.(s \models \heartsuit A \Longrightarrow t \models \heartsuit A).$$

Lemma 14. *Let $C = (X, \xi)$ and $D = (Y, \zeta)$ be T-coalgebras. A map $f : X \to Y$ is a Λ-homomorphism iff for all $x \in X$,*

$$Tf(\xi(x)) \leq_\Lambda \zeta(f(x)). \tag{8}$$

Proof. 'Only if': Let $\heartsuit \in \Lambda$, $A \subseteq Y$. Then

$$\begin{aligned} Tf(\xi(x)) \models \heartsuit A &\iff \xi(x) \models \heartsuit f^{-1}[A] && \text{(naturality)} \\ &\Longrightarrow \zeta(f(x)) \models \heartsuit f[f^{-1}[A]] && \text{(simulation)} \\ &\Longrightarrow \zeta(f(x)) \models \heartsuit A && \text{(monotony).} \end{aligned}$$

'If': Let $\xi(x) \models \heartsuit A$. We have to show $\zeta(f(x)) \models \heartsuit f[A]$, which will follow by (8) from $Tf(\xi(x)) \models \heartsuit f[A]$. By naturality, the latter is equivalent to $\xi(x) \models \heartsuit f^{-1}[f[A]]$. This however follows from $\xi(x) \models \heartsuit A$ by monotony. □

Remark 15. In the notation of the above lemma, another equivalent formulation of f being a Λ-homomorphism is that $\xi(x) \models \heartsuit f^{-1}[A]$ implies $\zeta(f(x)) \models \heartsuit A$ for $\heartsuit \in \Lambda$, $A \subseteq Y$. This is an immediate consequence of the lemma by naturality of predicate liftings.

As announced, Λ-simulations preserve the truth of positive modal formulas over Λ:

Theorem 16. *If S is a simulation and xSy, then $x \models_\xi \phi$ implies $y \models_\zeta \phi$ for every positive Λ-formula ϕ.*

Proof. Induction over ϕ, with trivial Boolean cases (noting that these do not include negation). For the modal case, we have

$$\begin{aligned} x \models_\xi \heartsuit\phi &\iff \xi(x) \models \heartsuit[\![\phi]\!] \\ &\implies \zeta(y) \models \heartsuit\{y' \mid \exists x'.(x' \models \phi \wedge x'Sy')\} \\ &\implies \zeta(y) \models \heartsuit[\![\phi]\!] \\ &\iff y \models_\zeta \heartsuit\phi. \end{aligned}$$

□

Example 17. 1. Over Kripke frames, when $\Lambda = \{\Diamond\}$, then a Λ-simulation $S : C \to D$ is just a simulation $C \to D$ in the usual sense. (Proof: 'only if': if xSy and $x' \in \xi(x)$, then $\xi(x) \models \Diamond\{x'\}$ and hence $\zeta(y) \models \Diamond\{y' \mid x'Sy'\}$, i.e. there exists y' such that $x'Sy'$ and $y' \in \zeta(y)$. 'If': If $\xi(x) \models \Diamond A$, then there exists $x' \in A \cap \xi(x)$ and hence we have $y' \in \zeta(y)$ such that $x'Sy'$, so that $\zeta(y) \models \Diamond\{y'' \mid \exists x'' \in \xi(x).\, x''Sy''\}$.)
2. When $\Lambda = \{\Box\}$, then a Λ-simulation $S : C \to D$ is just a simulation $D \to C$ in the usual sense. (Proof: 'only if': Let xSy and $y' \in \zeta(y)$. Assume that we cannot find $x' \in \xi(x)$ such that $x'Sy'$; that is, $\xi(x) \models \Box\{x' \mid \neg(x'Sy')\}$. Then by the definition of Λ-simulation, $\zeta(y) \models \Box A$ for an A with $y' \notin A$, contradiction. 'If': Let $\xi(x) \models \Box A$. To show that $\zeta(y) \models \Box\{y' \mid \exists x' \in A.\, x'Sy'\}$, let $y' \in \zeta(y)$. By the simulation property, there exists $x' \in \xi(x)$ such that $x'Sy'$, and since $\xi(x) \models \Box A$, we have $x' \in A$.)
3. Consequently, a $\{\Box, \Diamond\}$-simulation is a bisimulation in the usual sense.
4. For probabilistic modal logic, with $\Lambda = \{L_p \mid p \in [0,1] \cap \mathbb{Q}\}$, a relation $S \subseteq X \times Y$ between $\mathcal{D}$-coalgebras (X, ξ) and (Y, ζ) is a Λ-simulation iff for all xSy and all $A \subseteq X$,
$$\zeta(y)(S[A]) \geq \xi(x)(A)$$
(keep in mind that $\xi(x)$ and $\zeta(y)$ are probability measures that we can apply to subsets). The same comes out when we take $\Lambda = \{M_p \mid p \in [0,1] \cap \mathbb{Q}\}$.
5. For graded modal logic, with $\Lambda = \{\Diamond_k \mid k \in \mathbb{N}\}$, we obtain the same inequality characterizing Λ-simulations as for probabilistic logic (keeping in mind that we can see $\xi(x) \in \mathcal{B}_\infty(X)$, $\zeta(y) \in \mathcal{B}_\infty(Y)$ as discrete $\mathbb{N} \cup \{\infty\}$-valued measures).
6. For monotone neighbourhood logic, with $\Lambda = \{\Box\}$, we have that a relation $S \subseteq X \times Y$ between $\mathcal{M}$-coalgebras (X, ξ) and (Y, ζ) is a Λ-simulation iff for xSy, $A \in \xi(x)$ implies $S[A] \in \zeta(y)$.

For many purposes, simulations can be already too strong, e.g. when we are interested in preservation results for positive formulas up to a certain modal depth. It is therefore natural to consider n-simulations.

Definition 18 (Λ-n-simulation). Let $C = (X, \xi)$ and $D = (Y, \zeta)$ be T-coalgebras. We define the notion of *Λ-n-simulation* inductively as follows. Any $S_0 \subseteq X \times Y$ is a

Λ-0-simulation. A relation $S_{n+1} \subseteq X \times Y$ is a Λ-$(n+1)$-simulation if there exists a Λ-n-simulation S_n such that $S_{n+1} \subseteq S_n$ and for all x, y, $xS_{n+1}y$ implies that for all $\heartsuit \in \Lambda$, $A \subseteq X$

$$\xi(x) \models \heartsuit A \text{ implies } \zeta(y) \models \heartsuit S_n[A].$$

Theorem 19. *If S is a Λ-n-simulation and xSy, then $x \models_\xi \phi$ implies $y \models_\zeta \phi$ for every positive Λ-formula ϕ of rank at most n.*

Proof. Induction on n. The base case $n = 0$ is trivial since then ϕ is equivalent to either $\top$ or $\bot$. For $n > 0$, we proceed by induction on ϕ, the interesting case being:

$$\begin{aligned} x \models_\xi \heartsuit\psi &\iff \xi(x) \models \heartsuit[\![\psi]\!] \\ &\implies \zeta(y) \models \heartsuit S_{n-1}[[\![\psi]\!]] \\ &\implies \zeta(y) \models \heartsuit[\![\psi]\!] && \text{(outer IH + monotony)} \\ &\iff y \models_\zeta \heartsuit\psi. \end{aligned}$$

□

4 Bisimulations for All

The notion of Λ-(n)-simulation naturally yields a notion of bisimulation (i.e., simulations in both directions). The yardstick for any notion of bisimulation is *T-behavioural equivalence* (see Section 2). We say that a notion of bisimulation is *sound for T-behavioural equivalence* if any two states related by bisimulation are T-behaviourally equivalent, and *complete for T-behavioural equivalence* if any two T-behaviourally equivalent states can be related by a bisimulation.

The standard coalgebraic notion of (Aczel-Mendler) *T-bisimulation* that we recall below is always sound for T-behavioural equivalence, and complete for T-behavioural equivalence if T preserves weak pullbacks. We will show that our notion of Λ-bisimilarity is always sound and complete for T-behavioural equivalence, provided that Λ is separating. It has recently been shown that every finitary functor that preserves weak pullbacks admits a separating set of monotone predicate liftings [5], so that at least in the finitely branching case, Λ-bisimilarity applies (strictly) more widely than T-bisimilarity. Notice also that Λ-bisimulations enjoy nice closure properties, in particular under unions and composition (see Lemma 12), which for T-bisimulations is only the case, again, when T preserves weak pullbacks.

Definition 20. *Λ-bisimulation* is a Λ-simulation S such that S^{-1} is a Λ-simulation as well. Analogously, if S and its converse S^{-1} are Λ-n-simulations, then S is a *Λ-n-bisimulation*.

Example 21. 1. Over Kripke frames, both $\{\Box\}$-bisimulations and $\{\Diamond\}$-bisimulations are just bisimulations in the standard sense.
2. Using the description from Example 17, for probabilistic modal logic with $\Lambda = \{L_p \mid p \in [0,1] \cap \mathbb{Q}\}$, a relation $S \subseteq X \times Y$ between $\mathcal{D}$-coalgebras (X, ξ) and (Y, ζ) is a Λ-bisimulation iff for all xSy and all $A \subseteq X$, $B \subseteq Y$

$$\zeta(y)(S[A]) \geq \xi(x)(A) \quad \text{and} \quad \xi(x)(S^{-1}[A]) \geq \zeta(y)(A).$$

Note that standardly, probabilistic bisimulations are explicitly described (i.e. other than via Aczel-Mendler bisimulation) only for the case where S is an equivalence relation (see, e.g., [6,16]). In case $(X,\xi) = (Y,\zeta)$ and S is an equivalence relation on X, the condition above can be rewritten to obtain that S is a Λ-bisimulation iff for every set $A \subseteq X$ that is closed under S (so that $S[A] = A = S^{-1}[A]$),

$$\xi(x)(A) = \xi(y)(A)$$

whenever xSy, which is exactly the standard definition of probabilistic bisimulation for this case.

3. For $\Lambda = \{\Box\}$ interpreted over monotone neighbourhood frames, Λ-bisimulations are easily seen to be exactly monotone bisimulations as defined by Pauly [11] (see [3]; a similar definition for effectivity models can be found in [12, Definition 2.40]).

Lemma 22. *If C, D are T-coalgebras and $f : C \to D$ is a coalgebra morphism, then the graph of f is a Λ-bisimulation.*

Proof. It follows from Lemma 14 that the graph of f is a Λ-simulation. To see that its converse is a Λ-simulation, let $C = (X,\xi)$, $D = (Y,\zeta)$, and let $x \in X$, $\heartsuit \in \Lambda$, $A \subseteq Y$ such that $\zeta(f(x)) \models \heartsuit A$. Now $\zeta(f(x)) = Tf(\xi(x))$ because f is a coalgebra morphism, so we obtain $\xi(x) \models \heartsuit f^{-1}[A]$ by naturality of predicate liftings, as required. □

We are now ready to prove our main results stating that Λ-bisimulation is sound and complete for behavioural equivalence provided that Λ is separating.

Lemma 23. *The behavioural equivalence relation $\approx$ between two given T-coalgebras is a Λ-bisimulation.*

In other words, Λ-bisimulation is always complete for behavioural equivalence.

Proof. Let $C = (X,\xi)$, $D = (Y,\zeta)$ be T-coalgebras; it suffices to show that behavioural equivalence $\approx$ (as a relation between X and Y) is a Λ-simulation between C and D. Given $x \approx y$, $\heartsuit \in \Lambda$ and $A \subseteq X$ such that $\xi(x) \models \heartsuit A$, we then have to show that $\zeta(y) \models \heartsuit(\approx[A])$. So let E be a T-coalgebra and $f : C \to E$ and $g : D \to E$ be coalgebra morphisms such that $f(x) = g(y)$. By Lemma 22, and by stability of simulations under composition, the relation $g^{-1}f = \{(x',y') \mid f(x') = g(y')\}$ is a Λ-simulation. Thus, we have $\zeta(y) \models \heartsuit g^{-1}[f[A]]$; since $g^{-1}f$ is contained in $\approx$, we are done by monotony. □

Soundness depends, of course, on separation:

Theorem 24. *If Λ is separating, then Λ-bisimilarity is sound and complete for behavioural equivalence.*

Proof. As stated above, Lemma 23 proves completeness; it remains to show soundness. Let $C = (X,\xi)$ and $D = (Y,\zeta)$ be T-coalgebras, and let $S \subseteq X \times Y$ be a Λ-bisimulation. Let Z be the quotient of the disjoint sum $X + Y$ by the equivalence

relation generated by S, and let $\kappa_1 : X \to Z$ and $\kappa_2 : Y \to Z$ denote the prolongations of the coproduct injections into the quotient. It suffices to define a coalgebra structure χ on Z that makes κ_1 and κ_2 into coalgebra morphisms. We thus have to show that putting

$$\chi(\kappa_1(x)) = T\kappa_1(\xi(x))$$
$$\chi(\kappa_2(y)) = T\kappa_2(\zeta(x))$$

yields a well-defined map $Z \to TZ$. To this end, it suffices to show that $T\kappa_1(\xi(x)) = T\kappa_2(\zeta(y))$ whenever xSy. We prove this using separation by showing that $T\kappa_1(\xi(x)) \models \heartsuit A$ iff $T\kappa_2(\zeta(y)) \models \heartsuit A$ for $\heartsuit \in \Lambda$, $A \subseteq Z$. We prove only the left-to-right implication, the converse one being symmetric. So let $T\kappa_1(\xi(x)) \models \heartsuit A$. Then $\xi(x) \models \heartsuit \kappa_1^{-1}[A]$ by naturality, and hence $\zeta(y) \models \heartsuit S[\kappa_1^{-1}[A]]$ since S is a Λ-simulation. Now clearly $S[\kappa_1^{-1}[A]] \subseteq \kappa_2^{-1}[A]$, so that $\zeta(y) \models \heartsuit \kappa_2^{-1}[A]$ by monotony. We are done by naturality. □

As mentioned above, in the case where T preserves weak pullbacks, it is well-known that T-bisimilarity in the sense of Aczel and Mendler is also sound and complete for behavioural equivalence, so that T-bisimilarity and Λ-bisimilarity coincide when Λ is separating. But we can do better: T-bisimulations are Λ-bisimulations (so Λ-simulations are at least as convenient a tool as T-bisimulations), and for T preserving weak pullbacks and Λ separating, difunctional Λ-bisimulations are T-bisimulations. We recall the relevant definitions:

Definition 25. A *T-bisimulation* between T-coalgebras (X, ξ) and (Y, ζ) is a relation $S \subseteq X \times Y$ such that there exists a coalgebra structure $\rho : S \to TS$ that makes the projections $S \to X$ and $S \to Y$ into coalgebra morphisms.

Definition 26. A binary relation $S \subseteq X \times Y$ is *difunctional* if whenever xSy, zSy, and zSw, then xSw.

Essentially, we obtain a difunctional relation if we take an equivalence relation S on the disjoint union $X + Y$ of two sets and restrict it to $X \times Y$, i.e. take $S \cap (X \times Y)$ (where originally $S \subseteq (X + Y) \times (X + Y)$).

We now prove that all T-bisimulations are Λ-bisimulations, for any Λ and T, and that the converse holds for difunctional relations if T preserves weak pullbacks. We conjecture that the assumption of difunctionality can actually be removed. Moreover, we prove along the way that Λ-bisimulation up to difuctional closure is sound, a property that is then inherited by T-bisimulation in case T preserves weak pullbacks.

To begin, we note that every relation $S \subseteq X \times Y$ has a difunctional closure $\bar{S}$, where $x\bar{S}y$ iff there exists chains $x = x_0, \ldots, x_n$ in X and $y_0, \ldots, y_n = y$ in Y such that $x_i S y_i$ for $i = 0, \ldots, n$ and $x_{i+1} S y_i$ for $i = 0, \ldots, n-1$.

Definition 27. A *Λ-bisimulation up to difunctionality* between T-coalgebras (X, ξ) and (Y, ζ) is a relation $S \subseteq X \times Y$ such that whenever xSy and $\xi(x) \models \heartsuit A$ for $\heartsuit \in \Lambda$, $A \subseteq X$, then $\zeta(y) \models \heartsuit \bar{S}[A]$, where $\bar{S}$ denotes the difunctional closure of S, and the analogous condition holds for S^{-1}.

Proposition 28. *Let $S \subseteq X \times Y$ be a relation between T-coalgebras (X, ξ) and (Y, ζ). Then S is a Λ-bisimulation up to difunctionality iff the difunctional closure of S is a Λ-bisimulation.*

Proof. 'If' is trivial; we show 'only if'. Let $\bar{S}$ be the difunctional closure of S. Let $\heartsuit \in \Lambda$, $A \subseteq X$ such that $\xi(x) \models \heartsuit A$, and let $x\bar{S}y$, i.e. we have $x = x_0, \dots, x_n \in X$ and $y_0, \dots, y_n = y \in Y$ such that $x_i S y_i$ for $i = 0, \dots, n$ and $x_{i+1} S y_i$ for $i = 0, \dots, n-1$. We define $A_0, \dots, A_n \subseteq X$ and $B_0, \dots, B_n \subseteq Y$ inductively by $A_0 = A$, $B_i = \bar{S}[A_i]$, and $A_{i+1} = \bar{S}^{-1}[B_i]$. By induction, $\xi(x_i) \models \heartsuit A_i$ and $\zeta(y_i) \models \heartsuit B_i$ for all i. Moreover, by difunctionality of $\bar{S}$, $B_i = \bar{S}[A]$ for all i, so that $\zeta(y) = \zeta(y_n) \models \heartsuit \bar{S}[A]$ as required. The proof that $\bar{S}^{-1}$ is also a Λ-simulation is completely analogous. □

Corollary 29. *Let Λ be separating. Then Λ-bisimilarity up to difunctionality is sound and complete for T-behavioural equivalence.*

To complement this, we explicitly define a notion of T-bisimulation up to difunctionality:

Definition 30. A *T-bisimulation up to difunctionality* between T-coalgebras (X, ξ) and (Y, ζ) is a relation $S \subseteq X \times Y$ such that there exists a map $\rho : S \to T\bar{S}$, where $\bar{S}$ denotes the difunctional closure of S, such that $T\bar{p}_1 \rho = \xi p_1$ and $T\bar{p}_2 \rho = \zeta p_2$. Here $p_1 : S \to X$, $p_2 : S \to Y$, $\bar{p}_1 : \bar{S} \to X$, and $\bar{p}_2 : \bar{S} \to Y$ denote the projections.

It does not seem clear in general that an analogue of Proposition 28 holds for T-bisimulations. For the case where T preserves weak pullbacks, such an analogue will follow from the identification with Λ-bisimulations.

Theorem 31. *Every T-bisimulation (up to difunctionality) is a Λ-bisimulation (up to difunctionality).*

Proof. Let (X, ξ) and (Y, ζ) be T-coalgebras. The claim for plain bisimulations is immediate by Lemma 22 and stability of Λ-bisimulations under composition.

For the second part, let S be a T-bisimulation up to difunctionality between (X, ξ) and (Y, ζ), and let $\bar{S}$ denote the difunctional closure of S. Thus, we have $\rho : S \to T\bar{S}$ such that $T\bar{p}_1 \rho = \xi p_1$ and $T\bar{p}_2 \rho = \zeta p_2$, where $p_1 : S \to X$, $p_2 : S \to Y$, $\bar{p}_1 : \bar{S} \to X$, $\bar{p}_2 : \bar{S} \to Y$ denote the projections. Let $\heartsuit \in \Lambda$, $A \subseteq X$ such that $\xi(x) \models \heartsuit A$; we have to show $\zeta(y) \models \heartsuit \bar{S}[A]$. Now $\xi(x) = Tp_1\rho(x, y)$, and hence $\rho(x, y) \models \heartsuit \bar{p}_1^{-1}[A]$. Since $\zeta(y) = Tp_2\rho(x, y)$, we have to show $\rho(x, y) \models \heartsuit \bar{p}_2^{-1} S[A]$. By monotonicity, this follows from $\bar{p}_1^{-1}[A] \subseteq \bar{p}_2^{-1}[\bar{S}[A]]$. □

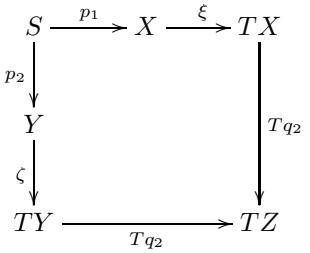
The announced partial converse to this is

Theorem 32. *If Λ is separating and T preserves weak pullbacks, then difunctional Λ-bisimulations are T-bisimulations, and Λ-bisimulations up to difunctionality are T-bisimulations up to difunctionality.*

Proof. For the first part, let $S \subseteq X \times Y$ be a difunctional Λ-bisimulation between T-coalgebras (X, ξ) and (Y, ζ). Let $p_1 : S \to X$ and $p_2 : S \to Y$ denote the projections. Let

$$\begin{array}{ccc} S & \xrightarrow{p_1} & X \\ {\scriptstyle p_2}\downarrow & & \downarrow{\scriptstyle q_2} \\ Y & \xrightarrow[q_2]{} & Z \end{array}$$

be a pushout; since S is difunctional, this is also a pullback. Now observe that the square



(9)

commutes. To show this, we use separation: let $\heartsuit \in \Lambda$, and let $A \subseteq Z$. After one application of naturality, we have to show that when xSy then $\xi(x) \models \heartsuit q_1^{-1}[A]$ iff $\zeta(x) \models \heartsuit q_2^{-1}[A]$. We show 'only if': observe that Z arises from $X + Y$ by quotienting modulo the equivalence relation $\sim_S$ generated by S. Thus $q_1^{-1}[A]$ consists of the elements of X whose $\sim_S$-equivalence class belongs to A, similarly for $q_2^{-1}[A]$. From $\xi(x) \models \heartsuit q_1^{-1}[A]$ we conclude $\zeta(y) \models \heartsuit S[q_1^{-1}[A]]$ because S is a Λ-simulation. But $S[q_1^{-1}[A]] \subseteq q_2^{-1}[A]$ because clearly each element of $S[q_1^{-1}[A]]$ is $\sim_S$-equivalent to an element of $q_1^{-1}[A]$ and hence its equivalence class belongs to A. Therefore, $\zeta(y) \models \heartsuit q_2^{-1}[A]$. The converse implication is shown dually.

From commutation of (9) we now obtain, by weak preservation of pullbacks, a map $\rho : S \to TS$ such that $Tp_1\rho = \xi p_1$ and $Tp_2\rho = \zeta p_2$, i.e. ρ is a witness for S being a T-bisimulation.

For the second part, let S be a Λ-bisimulation up to difunctionality. By Proposition 28, the difunctional closure $\bar{S}$ of S is a Λ-bisimulation and hence, by the first part, a T-bisimulation. By composing the T-coalgebra structure $\rho : \bar{S} \to T\bar{S}$ as in the definition of T-bisimulation with the inclusion $S \hookrightarrow \bar{S}$, we see that S is a T-bisimulation up to difunctionality. □

Corollary 33. *If T preserves weak pullbacks, then T-bisimulations up to difunctionality are sound (and complete) for T-behavioural equivalence.*

Similar results hold for Λ-n-bisimulation. It is easy to see that Λ-n-bisimulations preserve and reflect the truth of formulas with up to n nested modalities. A similar notion of preservation, *n-step-equivalence* was considered in [15], obtained by projecting into the terminal sequence. We can show that n-step-equivalence coincides with Λ-n-bisimilarity when Λ is separating. For the bounded depth version, no n-step variant of T-bisimulation seems to be available for comparison.

Definition 34. The *terminal sequence* of a given functor T is the sequence given by $T_0 = 1$ (some singleton set) and $T_{n+1} = TT_n$, connected by functions $p_n : T_{n+1} \to T_n$, where $p_{n+1} = Tp_n$. Every T-coalgebra $C = (X, \xi)$ defines a cone over the terminal sequence by $\xi_0 : C \to 1$ (uniquely defined) and $\xi_{n+1} = T\xi_n \circ \xi$. Given T-coalgebras (X, ξ) and (Y, ζ) and elements $x \in X$, $y \in Y$, we say that x and y are *n-step equivalent* (notation: $x \approx_n y$) whenever $\xi_n(x) = \zeta_n(y)$.

Lemma 35. *Let $C = (X, \xi)$ and $D = (Y, \zeta)$ be T-coalgebras. The n-step-equivalence relation $\approx_n \subseteq X \times Y$ is a Λ-n-bisimulation.*

Proof. Of course, it suffices to show that $\approx_n$ is a Λ-n-simulation. We proceed by induction on n. Clearly, $\approx_0 = X \times Y$ is a Λ-0-simulation. For the inductive step, let $x \approx_{n+1} y$ and let $\heartsuit \in \Lambda$, $A \subseteq X$ such that $\xi(x) \models \heartsuit A$. We then have

$$\begin{aligned}
\xi(x) \models \heartsuit A &\implies \xi(x) \models \heartsuit \xi_n^{-1}[\xi_n[A]] && \text{(monotony)}\\
&\implies T\zeta_n(\zeta(y)) = \zeta_{n+1}(y) = \xi_{n+1}(x) && (x \approx_{n+1} y)\\
&\qquad = (T\xi_n)(\xi(x)) \models \heartsuit \xi_n[A] && \text{(naturality)}\\
&\implies \zeta(y) \models \heartsuit \zeta_n^{-1}[\xi_n[A]] && \text{(naturality)}\\
&\qquad = \heartsuit \approx_n [A] && \text{(definition of } \approx_n).
\end{aligned}$$

By the inductive hypothesis, $\approx_n$ is a Λ-n-simulation, and, moreover, $\approx_{n+1} \subseteq \approx_n$, so $\approx_{n+1}$ is a Λ-$(n+1)$-simulation. □

Of course, the converse of this lemma does not hold in general (e.g., take T to be the multiset functor and consider $\Lambda = \{\Diamond_0\}$). However, we do have the following.

Theorem 36. *If Λ is a separating set of predicate liftings, then $S_n \subseteq \approx_n$ for every Λ-n-bisimulation S_n.*

Proof. Induction on n. Let $C = (X, \xi)$, $D = (Y, \zeta)$ be T-coalgebras, let $S_{n+1} \subseteq X \times Y$ be a Λ-$(n+1)$-bisimulation, and let $xS_{n+1}y$. Let $S_n \supseteq S_{n+1}$ be an n-bisimulation as in the definition of Λ-$(n+1)$-bisimilarity.

We show $\xi_{n+1}(x) = \zeta_{n+1}(y)$ using separation. Thus, let $\heartsuit \in \Lambda$, $A \subseteq T_n$. We have to show that $\xi_{n+1}(x) \models \heartsuit A$ iff $\zeta_{n+1}(y) \models \heartsuit A$; by symmetry, it suffices to prove 'only if'. Since $\xi_{n+1} = T\xi\xi_n$, we have, by naturality, $\xi(x) \models \heartsuit \xi_n^{-1}[A]$. By simulation, it follows that $\zeta(y) \models \heartsuit S_n[\xi_n^{-1}[A]]$. By the inductive hypothesis, $S_n \subseteq \approx_n$, so that we obtain $\zeta(y) \models \heartsuit \approx_n [\xi_n^{-1}[A]]$ by monotony. Now $\approx_n [\xi_n^{-1}[A]] = \zeta_n^{-1}[A]$ by definition of $\approx_n$, and hence $\zeta_{n+1}(y) = T\zeta_n\zeta(y) \models \heartsuit A$ by naturality. □

In other words, Λ-n-bisimulation is always complete for n-step equivalence, and sound if Λ is separating.

5 Related Work

Recent work by Enqvist [2] introduces a notion of *Λ-homomorphism* that is almost a special case of a Λ-simulation, and in fact shows that such Λ-homomorphisms can be induced by a relator in the sense of [7], so that a notion related to Λ-simulation can itself be regarded as implicit in that work. When we say 'almost', we mean that the implication in the definition of Λ-homomorphism goes the other way in Enqvist's work than it does here, so that in particular Theorem 16 would fail for his notion.

In [8] it is shown that so-called *lax extensions of T preserving diagonals* induce notions of bisimulation that are sound and complete for behavioural equivalence, and that a finitary functor has such an extension iff it admits a separating set of finitary monotone predicate liftings. Our result, while otherwise working with similar assumptions, does not suppose finitariness of the functor.

In [7] a generic theory of coalgebraic simulation is developed using *relators*. One can show that our notion of Λ-simulation is induced by a relator and therefore subsumed by that framework. We cannot currently make out that any of our results about Λ-(bi)simulation could be obtained by instantiating the generic results, however.

6 Conclusions

We have introduced novel notions of Λ-simulation and Λ-bisimulation that work well in a setting where the coalgebraic type functor admits a separating set Λ of monotone predicate liftings. In particular, we have shown that Λ-bisimilarity is, in this setting, always sound and complete for T-behavioural equivalence, and moreover always admits a natural notion of bisimulation up to difunctionality. We have shown that T-bisimulations are always Λ-bisimulations, similarly for versions up to difunctionality, and that the converse holds for versions up to difunctionality in case T preserves weak pullbacks. We leave the question whether the converse holds in the plain case under preservation of weak pullbacks as an open problem.

Acknowledgements. We wish to thank Helle Hvid Hansen for useful pointers to the literature. Erwin R. Catesbeiana has voiced vague support for empty bisimulations.

References

1. Barr, M.: Terminal coalgebras in well-founded set theory. Theoret. Comput. Sci. 114, 299–315 (1993)
2. Enqvist, S.: Homomorphisms of coalgebras from predicate liftings. In: Heckel, R., Milius, S. (eds.) CALCO 2013. LNCS, vol. 8089, pp. 126–140. Springer, Heidelberg (2013)
3. Hansen, H., Kupke, C.: A coalgebraic perspective on monotone modal logic. In: Coalgebraic Methods in Computer Science (CMCS 2004). ENTCS, vol. 106, pp. 121–143. Elsevier (2004)
4. Hennessy, M., Milner, R.: On observing nondeterminism and concurrency. In: de Bakker, J.W., van Leeuwen, J. (eds.) ICALP 1980. LNCS, vol. 85, pp. 299–309. Springer, Heidelberg (1980)
5. Kurz, A., Leal, R.A.: Modalities in the Stone age: A comparison of coalgebraic logics. Theor. Comput. Sci. 430, 88–116 (2012)
6. Larsen, K., Skou, A.: Bisimulation through probabilistic testing. Inf. Comput. 94, 1–28 (1991)
7. Levy, P.B.: Similarity quotients as final coalgebras. In: Hofmann, M. (ed.) FOSSACS 2011. LNCS, vol. 6604, pp. 27–41. Springer, Heidelberg (2011)
8. Marti, J., Venema, Y.: Lax extensions of coalgebra functors. In: Pattinson, D., Schröder, L. (eds.) CMCS 2012. LNCS, vol. 7399, pp. 150–169. Springer, Heidelberg (2012)
9. Pattinson, D.: Coalgebraic modal logic: Soundness, completeness and decidability of local consequence. Theoret. Comput. Sci. 309, 177–193 (2003)
10. Pattinson, D.: Expressive logics for coalgebras via terminal sequence induction. Notre Dame J. Formal Logic 45, 2004 (2002)
11. Pauly, M.: Bisimulation for general non-normal modal logic (1999) (unpublished Manuscript)

12. Pauly, M.: Logic for social software. Ph.D. thesis, Universiteit van Amsterdam (2001)
13. Schröder, L., Pattinson, D.: PSPACE bounds for rank-1 modal logics. ACM Trans. Comput. Log. 10, 13:1–13:33 (2009)
14. Schröder, L.: Expressivity of coalgebraic modal logic: The limits and beyond. Theoret. Comput. Sci. 390(2-3), 230–247 (2008)
15. Schröder, L., Pattinson, D.: Coalgebraic correspondence theory. In: Ong, L. (ed.) FOSSACS 2010. LNCS, vol. 6014, pp. 328–342. Springer, Heidelberg (2010)
16. Sokolova, A.: Probabilistic systems coalgebraically: A survey. Theoret. Comput. Sci. 412, 5095–5110 (2011)

A Coalgebraic View of ε-Transitions

Alexandra Silva* and Bram Westerbaan

ICIS, Radboud University Nijmegen

Abstract. In automata theory, a machine transitions from one state to the next when it reads an input symbol. It is common to also allow an automaton to transition without input, via an ε-transition. These ε-transitions are convenient, e.g., when one defines the composition of automata. However, they are not necessary, and can be eliminated. Such ε-elimination procedures have been studied separately for different types of automata, including non-deterministic and weighted automata.

It has been noted by Hasuo that it is possible to give a coalgebraic account of ε-elimination for some automata using trace semantics (as defined by Hasuo, Jacobs and Sokolova).

In this paper, we give a detailed description of the ε-elimination procedure via trace semantics (missing in the literature). We apply this framework to several types of automata, and explore its boundary.

In particular, we show that is possible (by careful choice of a monad) to define an ε-removal procedure for all weighted automata *over the positive reals* (and certain other semirings). Our definition extends the recent proposals by Sakarovitch and Lombardy for these semirings.

1 Introduction

Automata are among the most basic structures in Computer Science. They have applications in a wide range of areas, including parsing, speech processing, and image recognition/generation software. Despite their simplicity, much research is still devoted to the semantics of automata and of related constructions.

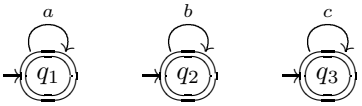
Coalgebra is a mathematical framework to study dynamical systems, of which automata are prime examples. Deterministic automata were the first automata to be studied as coalgebras in the seminal paper by Rutten [15]. Subsequently, various other types of automata and constructions were studied coalgebraically. This view has unified and generalized existing results and algorithms for different types of automata [17,1,2,18,3].

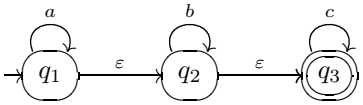
In this paper, we give a coalgebraic account of another concrete construction for automata: the elimination of ε-transitions. For this we use the abstract machinery of trace semantics. The advantage of this combination is two-fold. On the one hand, the concrete examples that the various types of automata provide clarify and ground the abstract notion of trace. On the other hand, trace semantics provides us with a uniform and intuitive definition for ε-elimination for many types of automata.

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

R. Heckel and S. Milius (Eds.): CALCO 2013, LNCS 8089, pp. 267–281, 2013.

ε-Transitions are often useful at an intermediate stage. To illustrate this, let us show how to construct a non-deterministic automaton (without ε-transitions) that recognizes the language $a^*b^*c^*$. Note that it is easy to find automata recognizing the languages a^*, b^* and c^* (above, respectively).

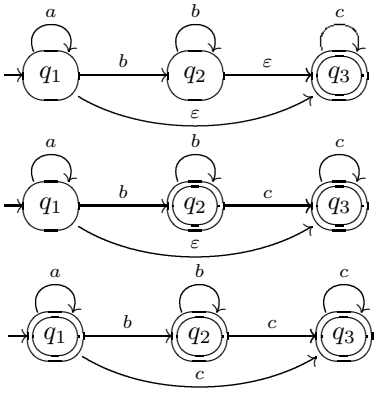




If we compose these automata using ε-transitions, we obtain an automaton, on the left, that recognizes $a^*b^*c^*$. To obtain an automaton *without* ε-transitions that recognizes $a^*b^*c^*$ we incrementally eliminate the ε-transitions, as displayed below.

Hasuo and others [7,9] noted that result of the iterative process seen above can be captured using trace semantics in a Kleisli category, approach which we will discuss in more detail in Section 2.

In this paper, we take inspiration from [7,9] and we give an elaborate treatment of ε-elimination procedures using trace semantics. We extend their theory to include a class of weighted automata.



Though the process of ε-elimination for non-deterministic automata is classical and well-understood, for weighted automata things are less clear-cut, as witnessed by recent research [14,12]. The construction presented in this paper brings new results in comparison with the research presented in the aforementioned papers.

From a coalgebraic perspective, the challenge behind ε-elimination comes from the fact that many notions and definitions, such as bisimilarity for a functor, are given in a step-wise fashion. That is, the behavior of a certain system is fully determined by looking one step ahead at each time. This phenomenon, of having to deal with *multi-step* behavior, poses problems when having to model internal actions, such as ε-transitions, of a system. This is also present in concurrency theory, where internal actions (τ-steps) are discarded when defining weak bisimilarity. The theory presented in this paper might give a direction to improve the existing coalgebraic accounts of weak bisimilarity [20,4], which are not yet satisfactory.

The paper is organized as follows. In Section 2, we discuss the concrete construction for non-deterministic automata, we discuss how this paves the way to a coalgebraic account, and we introduce the idiosyncrasies behind the analogous construction for weighted automata. In Section 3, we present the general framework to formalize elimination of ε-transitions. In Section 4, we show how to model weighted automata in order to fit the framework. In Section 5, we discuss directions for future work.All proofs are omitted in the present article. A full version, containing all proofs and extra material can be found in [19].

2 Motivation

In this section we describe the existing ε-elimination procedures for weighted and non-deterministic automata more thoroughly. We also recall some of the basic notions concerning automata. We present the material in a manner that is suited to the purposes of this paper. For instance, we represent these automata as coalgebras and make no mention of initial states.

2.1 Non-deterministic Automata

We represent a **non-deterministic automaton** (NDA) with states X over the alphabet A as a map $\alpha\colon X \longrightarrow \wp(A \times X + 1)$, where $\wp$ is the powerset functor. Given $q, r \in X$ and $a \in A$, we write, omitting the coproduct injections,

$$\begin{array}{lcll} q{\downarrow_\alpha} & \iff & * \in \alpha(q) & q \text{ is a final state} \\ q \xrightarrow{a}_\alpha r & \iff & (a, r) \in \alpha(q) & q \text{ has an } a\text{-transition to } r \end{array}$$

Let us recall the usual (language) semantics of α, i.e., which words a state $q \in X$ of α *accepts*. Let $w \equiv a_1 a_2 \cdots a_n$ be a word over A, and let $q \in X$. We say that q **accepts** w if there are $q_1, \ldots, q_n \in X$ such that

$$q \xrightarrow{a_1}_\alpha q_1 \xrightarrow{a_2}_\alpha \cdots \xrightarrow{a_n}_\alpha q_n \quad \text{and} \quad q_n{\downarrow_\alpha}. \tag{1}$$

So the semantics of α is captured by the map $[\![-]\!]_\alpha\colon X \longrightarrow \wp(A^*)$ given by

$$w \in [\![q]\!]_\alpha \quad \iff \quad q \text{ accepts } w,$$

where $q \in X$ and $w \in A^*$. So we will simply say that $[\![-]\!]_\alpha$ *is* the semantics of α.

ε-Transitions An NDA **with ε-transitions** (ε-NDA) with states X over an alphabet A is simply an NDA with states X over the alphabet $A + \{\varepsilon\}$,

$$\alpha\colon X \longrightarrow \wp((A + \{\varepsilon\}) \times X + 1),$$

but with a different semantics, which we define next.

Given a word $\tilde{w}$ over $A + \{\varepsilon\}$, let $\tilde{w}\backslash\varepsilon$ be the word on A one obtains by removing all the letters "ε" from $\tilde{w}$.

Let $w \in A^*$, and let $q \in X$. We say q **accepts** w (in the ε-NDA α) if there is $\tilde{w} \in (A + \{\varepsilon\})^*$ such that $w = \tilde{w}\backslash\varepsilon$ and q accepts $\tilde{w}$ in α seen as an NDA, as in (1).

Hence the semantics of α is the map $[\![-]\!]^\varepsilon_\alpha\colon X \longrightarrow \wp(A^*)$ given by, for $q \in X$,

$$[\![q]\!]^\varepsilon_\alpha \;=\; \{\, \tilde{w}\backslash\varepsilon\colon \;\; \tilde{w} \in [\![q]\!]_\alpha \,\}.$$

Or, more abstractly, $[\![-]\!]^\varepsilon_\alpha = \wp(-\backslash\varepsilon) \circ [\![-]\!]_\alpha$.

ε-Elimination for Non-deterministic Automata. Let α be an ε-NDA with states X and over an alphabet A. We construct an NDA $\alpha^{\#} \colon X \to \wp(A \times X + 1)$ which has the same semantics as α, in the sense that $[\![-]\!]_{\alpha^{\#}} = [\![-]\!]^{\varepsilon}_{\alpha}$. Since $\alpha^{\#}$ will have no ε-transitions we say "we have eliminated the ε-transitions".

The NDA $\alpha^{\#}$ is defined as follows. A state $q \in X$ has a transition in $\alpha^{\#}$ labelled by $a \in A$ to a state r if either this transition was already there in α or after a number of ε-transitions, starting from q, it is possible to make an a-transition to r. Formally:

$$q{\downarrow_{\alpha^{\#}}} \iff \left[\begin{array}{l} q \xrightarrow{\varepsilon}_{\alpha} q_1 \xrightarrow{\varepsilon}_{\alpha} \cdots \xrightarrow{\varepsilon}_{\alpha} q_n \quad \text{and} \quad q_n{\downarrow_{\alpha}} \\ \quad \text{for some } n \in \mathbb{N} \text{ and } q_1, \ldots, q_n \in X \end{array} \right.$$

$$q \xrightarrow{a}_{\alpha^{\#}} r \iff \left[\begin{array}{l} q \xrightarrow{\varepsilon}_{\alpha} q_1 \xrightarrow{\varepsilon}_{\alpha} \cdots \xrightarrow{\varepsilon}_{\alpha} q_n \quad \text{and} \quad q_n \xrightarrow{a}_{\alpha} r \\ \quad \text{for some } n \in \mathbb{N} \text{ and } q_1, \ldots, q_n \in X \end{array} \right.$$

Let $w \in A^*$ and $q \in X$. We leave it to the reader to verify that q accepts w in the ε-NDA α if and only if q accepts w in the NDA $\alpha^{\#}$, i.e., $[\![q]\!]_{\alpha^{\#}} = [\![q]\!]^{\varepsilon}_{\alpha}$. Hence, the following diagram commutes.

$$\begin{array}{ccc} X & \xrightarrow{[\![-]\!]_{\alpha}} & \wp((A + \{\varepsilon\})^*) \\ & \searrow^{[\![-]\!]_{\alpha^{\#}}} & \downarrow{\scriptstyle \wp(-\setminus\varepsilon)} \\ & & \wp(A^*) \end{array}$$

Note that $[\![-]\!]_{\alpha}$ is the semantics of α considered as an NDA.

Coalgebraic Formulation. We want to find an abstract definition of $\alpha^{\#}$ so that it can be instantiated for other types of automata. To this end it turns out to be fruitful to consider the following variant of $\alpha^{\#}$. Let

$$\mathrm{tr}_{\alpha} \colon X \longrightarrow \wp(\, \mathbb{N} \times (A \times X + 1)\,)$$

be the map given by: for all $q, r \in X$, and $a \in A$, and $n \in \mathbb{N}$:

$$(n, (a, r)) \in \mathrm{tr}_{\alpha}(q) \iff \left[\begin{array}{l} q \xrightarrow{\varepsilon}_{\alpha} q_1 \xrightarrow{\varepsilon}_{\alpha} \cdots \xrightarrow{\varepsilon}_{\alpha} q_n \quad \text{and} \quad q_n \xrightarrow{a}_{\alpha} r \\ \quad \text{for some } q_1, \ldots, q_n \in X \end{array} \right.$$

$$(n, *) \in \mathrm{tr}_{\alpha}(q) \iff \left[\begin{array}{l} q \xrightarrow{\varepsilon}_{\alpha} q_1 \xrightarrow{\varepsilon}_{\alpha} \cdots \xrightarrow{\varepsilon}_{\alpha} q_n \quad \text{and} \quad q_n{\downarrow_{\alpha}} \\ \quad \text{for some } q_1, \ldots, q_n \in X \end{array} \right.$$

The map tr_{α} contains more information than $\alpha^{\#}$. For example, $\alpha^{\#}$ tells us if a state $q \in X$ is final by whether $* \in \alpha^{\#}(q)$. The map tr_{α} tells us more, namely whether a final state can be reached from the state q using exactly n ε-transitions by whether $(n, *) \in \mathrm{tr}_{\alpha}(q)$.

Note that we can recover $\alpha^{\#}$ from the map tr_{α}; we have

$$b \in \alpha^{\#}(q) \iff \exists n \in \mathbb{N} \quad (n, b) \in \mathrm{tr}_{\alpha}(q), \tag{2}$$

for all $q \in X$ and $b \in B$, where $B := A \times X + 1$. More categorically, we can formulate Statement (2) as:

$$X \xrightarrow{\mathrm{tr}_\alpha} \wp(\,\mathbb{N} \cdot B\,) \xrightarrow{\wp(\nabla)} \wp(B) \qquad \text{commutes.}$$
$$\alpha^{\#}$$

Here, $\mathbb{N} \cdot B$ is the countable coproduct and $\nabla \colon \mathbb{N} \cdot B \to B$ is the *codiagonal* given by $\nabla(n, b) = b$ for all $(n, b) \in \mathbb{N} \cdot B$.

We are interested in tr_α because it satisfies a recursive relation, namely

$$\begin{aligned} (0, b) \in \mathrm{tr}_\alpha(q) \quad &\iff \quad b \in \alpha(q) \\ (n+1, b) \in \mathrm{tr}_\alpha(q) \quad &\iff \quad \exists r \in X \big[\, (\varepsilon, r) \in \alpha(q) \;\wedge\; (n, b) \in \mathrm{tr}_\alpha(r) \,\big], \end{aligned} \tag{3}$$

where $q \in X$ and $n \in \mathbb{N}$ and $b \in B$.

The recursive relation (3) can be cast in an abstract form, and this allows us to define tr_α (and hence $\alpha^{\#}$) for different types of automata at once.

For this we will use the Kleisli category $\mathcal{K}\ell(\wp)$ of the monad $\wp$. Recall that a map $f \colon V \to \wp(W)$ is a morphism from V to W in $\mathcal{K}\ell(\wp)$, which we will write as $f \colon\; V \relbar\!\!\circ\!\!\rightarrow W$.

Indeed, we will see that the map $\mathrm{tr}_\alpha \colon X \to \wp(\mathbb{N} \cdot B)$ is the unique morphism such that the following diagram commutes, in $\mathcal{K}\ell(\wp)$.

$$\begin{array}{ccc} X & \xrightarrow{\mathrm{tr}_\alpha} & \mathbb{N} \cdot B \\ {\scriptstyle \alpha'}\downarrow & & \downarrow{\scriptstyle \xi} \\ X + B & \xrightarrow[\mathrm{tr}_\alpha + 1]{} & \mathbb{N} \cdot B + B \end{array} \quad ,$$

where $\alpha' \colon X \longrightarrow \wp(X + B)$ is the composition of the following maps

$$X \xrightarrow{\alpha} \wp(\,(A + \{\varepsilon\}) \times X + 1\,) \xrightarrow{\cong} \wp(\,X + (A \times X + 1)\,)\,, \tag{4}$$

and $\xi \colon \mathbb{N} \cdot B \longrightarrow \wp(\mathbb{N} \cdot B + B)$ is given by $\xi(0, b) = \{b\}$, and $\xi(n+1, b) = \{(n, b)\}$, for all $b \in B$ and $n \in \mathbb{N}$.

We can formulate this more coalgebraically, as follows. Let F be the functor on $\mathcal{K}\ell(\wp)$ given by $F = - + B$. Then we can regard α' as an F-coalgebra,

$$X \xrightarrow{\alpha'} FX \;= X + B$$

The final F-coalgebra is ξ, and tr_α is the unique homomorphism from α' to ξ.

Such a unique homomorphism tr_α into the final coalgebra in a Kleisli category is called a *trace map* by Hasuo, Jacobs, and Sokolova [8].

We will use the observations above to study ε-elimination in a more general setting later on. But let us first consider the class of weighted automata.

2.2 Weighted Automata

Let S be a semiring (such as $\mathbb{R}$). A weighted automaton is similar to a non-deterministic automaton, but each transition and state carries a weight, $s \in S$.

Depending on the semiring, one may think of the weight of the transition between two states q and r as the distance of the transition from q to r, or as the probability that α transitions from q to r. For more information on weighted automata, see [5].

We represent a **weighted automaton** over the semiring S with states X over an alphabet A by a map $\alpha\colon X \longrightarrow \mathscr{M}(A \times X + 1)$, where $\mathscr{M}$ is the *multiset monad* over S. Recall that

$$\mathscr{M}(X) \;=\; \big\{\varphi \mid \varphi\colon X \to S,\ \operatorname{supp}\varphi \text{ is finite } \big\}.$$

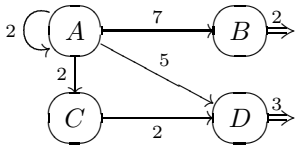
Given $q, r \in X$ and $a \in A$ and $s \in S$, we write

$$\begin{aligned} q{\downarrow}^{s}_{\alpha} \quad &\iff \quad s \;=\; \alpha(q)(*) && q \text{ outputs weight } s \\ q \xrightarrow{a|s}_{\alpha} r \quad &\iff \quad s \;=\; \alpha(q)(a,r) && q \text{ has an } a\text{-transition to } r \text{ with weight } s \end{aligned}$$

The subscript α will be omitted whenever it is clear from the context.

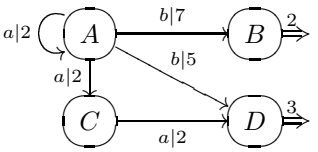
Let $a \in A$ be given. Note that formally there is an a-transition between any pair of states with *some* weight. We will typically only depict transitions with non-zero weight.

Semantics. We explain the semantics of weighted automata by an example. Consider the following variation on a directed graph that represents a maze.



Suppose we stand at vertex A, and want to find the shortest path to exit the maze (via one of the exits, $\Rightarrow$). It is $A \to C \to D \Rightarrow$ with length 7.

Let us increase the complexity of the maze by adding some labels.



(5)

Again, we stand at A and want to find the shortest path to one of the exits, but this time we are only allowed to move along an ab-labelled path. That is, to exit the maze, we are only allowed to first move along an edge labelled by a, and then along an edge labelled by b, and then along $\Rightarrow$. Now the shortest path is $A \to A \to D \Rightarrow$ with length 10.

The maze in (5) can be represented by a weighted automaton α with states $X := \{A, B, C, D\}$ and alphabet $A := \{a, b\}$ over a semiring[1] on $\mathbb{R} \cup \{+\infty\}$

[1] The appropriate semiring structure on $\mathbb{R} \cup \{+\infty\}$ will become clear later on.

in a straightforward manner. When there is no c-labelled edge from one vertex to another we use a c-transition of weight $+\infty$, e.g., $\alpha(B)(a, D) = +\infty$. We interpret the symbol " $\overset{s}{\Rightarrow}$ " at a vertex q to mean that q outputs s.

Note that we can express the length of the shortest ab-labelled path from A to an exit using α as follows.

$$\min_{q_1 \in X} \min_{q_2 \in X} \Big[\; \alpha(A)(\kappa_\ell(a, q_1)) \; + \; \alpha(q_1)(\kappa_\ell(b, q_2)) \; + \; \alpha(q_2)(\kappa_r(*)) \; \Big]$$

Note that "+" and "min" form a semiring $\mathscr{T}_{\min}$ on $\mathbb{R} \cup \{+\infty\}$, called the *tropical semiring*. Confusingly, "+" is the *multiplication* of $\mathscr{T}_{\min}$ while "min" is the *addition*. Hence the *zero* of $\mathscr{T}_{\min}$ is $+\infty$ and the *one* is 0.

Observe that if we change the operations "+" and "min" (that is, if we change the semiring on $\mathbb{R} \cup \{+\infty\}$) we get different semantics $[\![-]\!]_\alpha$. For instance, if we take "+" and "max" instead, $[\![q]\!]_\alpha(w)$ will be the the length of the *longest* w-labelled path from q to an exit.

We now give the general definition of semantics for weighted automata. Let S be a semiring. Let $\alpha \colon X \longrightarrow \mathscr{M}(A \times X + 1)$ be a weighted automaton over S. Then the **semantics** of α is the map $[\![-]\!]_\alpha \colon X \longrightarrow S^{A^*}$, given by, for $q_1 \in X$, and a word $w = a_1 \cdots a_n \in A^*$,

$$[\![q_1]\!]_\alpha(w) \; := \sum_{q_2 \in X} \cdots \sum_{q_{n+1} \in X} \Big(\prod_{i=1}^{n} \alpha(q_i)(a_i, q_{i+1}) \Big) \cdot \alpha(q_{n+1})(*). \tag{6}$$

So a state in the weighted automaton α recognizes functions in S^{A^*}. These functions are usually referred to as *formal power series (over S)*.

Non-deterministic automata are a special case of weighted automata. Indeed, the reader can verify that if we take S to be the Boolean semiring then weighted automata over S correspond exactly to NDAs.

ε-Transitions. A **weighted automaton with ε-transitions** α over a semiring S with states X and alphabet A is simply a weighted automaton over S with states X and alphabet $A + \{\varepsilon\}$.

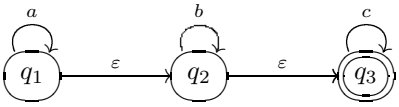
To explain the semantics of α, we first consider the tropical case $S = \mathscr{T}_{\min}$. Following the earlier discussion of the semantics of ordinary weighted automata over $\mathscr{T}_{\min}$ and shortest paths, it seems natural to define the semantics of α to be the map $[\![-]\!]^\varepsilon_\alpha \colon X \longrightarrow S^{A^*}$ given by, for $q \in X$ and $w \in A^*$,

$$[\![q]\!]^\varepsilon_\alpha(w) \;=\; \min \big\{ \, [\![q]\!]_\alpha(\tilde{w}) \colon \; \tilde{w} \in (A + \{\varepsilon\})^* \text{ and } \tilde{w} \backslash \varepsilon = w \, \big\}. \tag{7}$$

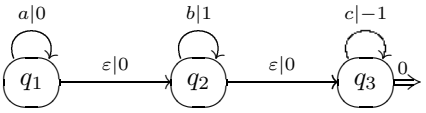
In the maze analogy, $[\![q]\!]^\varepsilon_\alpha(w)$ is the length of a shortest w-labelled path from q to an exit when ε-moves are not counted.

However, note that Equation (7) is not a sound definition for all α since the minimum might not exist. We will return to this problem shortly.

But first, we will further illustrate the semantics of ε-transitions. Recall that state q_1 in the following NDA accepts the language denoted by $a^*b^*c^*$:



Instead of talking about acceptance we now want to assign to each word in the language $a^*b^*c^*$ the difference between the number of b's and c's occurring in the word. In order to do that, we modify the above automaton into a weighted automaton over the tropical semiring $\mathscr{T}_{\min}$.



Note that for $w \in \{a, b, c\}^*$ the weight $[\![q_1]\!]^{\varepsilon}_{\alpha}(w)$ is precisely the number of b's occuring in w minus the number of c's occuring in w.

Inspired by Equation (7) we would like to define the semantics of a weighted automaton α with ε-transitions over any semiring S with states X and alphabet A to be the map $[\![-]\!]^{\varepsilon}_{\alpha} \colon X \longrightarrow S^{A^*}$ given by, for $q \in X$ and $w \in A^*$,

$$[\![q]\!]^{\varepsilon}_{\alpha}(w) \;=\; \sum \left\{\, [\![q]\!]_{\alpha}(\tilde{w}) \colon\; \tilde{w} \in (A + \{\varepsilon\})^* \text{ and } \tilde{w}\backslash\varepsilon = w \,\right\}. \tag{8}$$

However, without further information this is only a valid definition if the set

$$\left\{\, [\![q]\!]_{\alpha}(\tilde{w}) \colon\; \tilde{w} \in (A + \{\varepsilon\})^* \text{ and } \tilde{w}\backslash\varepsilon = w \,\right\}. \tag{9}$$

is finite. Otherwise, we do not know how we should interpret the symbol "$\sum$".

The problem is quite subtle. For example, consider the following weighted automata with ε-transitions over $\mathbb{R}$ (with the normal "+" and "$\cdot$").

Writing $\square$ for the empty word, one sees using Equation (8), that

$$\begin{aligned} [\![q_1]\!]^{\varepsilon}(\square) \;&=\; 1 + 0.5 + (0.5)^2 + \cdots \;=\; 2, \\ [\![q_2]\!]^{\varepsilon}(\square) \;&=\; 1 - 0.5 + (0.5)^2 - \cdots \;=\; 2/3, \end{aligned}$$

and in a daring mood we can compute,

$$[\![q_3]\!]^{\varepsilon}(\square) \;=\; 1 + 2 + 4 + 8 + \cdots \;=\; +\infty,$$

but what should we make of the following?

$$[\![q_4]\!]^{\varepsilon}(\square) \;=\; 1 - 1 + 1 - 1 + \cdots$$

To give proper meaning to weighted automata with ε-transitions it seems necessary to require that the semiring is equipped with a notion of summation for some sequences, and we must restrict ourselves to a class of weighted automata with ε-transitions for which the set in Expression (9) is summable.

Possibly due to this problem, the formal semantics of weighted automata with ε-transitions has not yet been settled in the literature.

In a recent proposal by Lombardy and Sakarovitch [12], semantics is given to a certain class of '*valid*' weighted automata with ε-transitions over *topological* semirings using a sophisticated ε-elimination *algorithm*. The automata with states q_1 and q_2 are valid, and the other two are not valid.

In this paper, the abstract view on automata gives rise to semantics to *all* weighted automata over certain semirings, namely *positive partial σ-semirings*. The semiring $[0, +\infty)$ is such a semiring, while $\mathbb{R}$ is not. So the general theory yields semantics for the automata with states q_1 and q_3, but not for the automata with states q_2 and q_4.

We will return to the example of weighted automata is Section 4.

3 Generalised ε-Elimination

Let us now turn to ε-elimination in a more general setting.

3.1 Automata in General

Setting 1. *Let* $\mathbf{C}$ *be a category, and assume that* $\mathbf{C}$ *has all finite limits and all countable colimits. Let* F *be a functor on* $\mathbf{C}$*, and let* T *be a monad on* $\mathbf{C}$*, with Kleisli category* $\mathcal{K}\ell(T)$*.*

In this setting, we abstractly define an automaton, parametrized by a functor F and a monad T, as follows.

Definition 2. *Let* X *be an object from* $\mathbf{C}$*. An* ***automaton*** *of type* T, F *with states* X *is a morphism* $\alpha\colon X \longrightarrow TFX$*.*

In other words, an automaton of type T, F is a morphism in $\mathcal{K}\ell(T)$ of the form

$$\alpha\colon X \longrightarrow FX.$$

Examples 3. *Let* $\mathbf{C} = \mathbf{Sets}$*, and* $F = A \times - + 1$ *for some object* A *of* $\mathbf{C}$*.*

(i) Let $T = \wp$ *be the powerset monad. Then the automata of type* T, F *are non-deterministic automata with alphabet* A*.*

(ii) Let S *be a semiring. Let* $T := \mathscr{M}$ *be the multiset monad over* S*. Then the automata of type* T, F *are weighted automata with alphabet* A *over* S*.*

Example 4. *Let* $\mathbf{C} = \mathbf{Meas}$*. Let* $F = A \times - + 1$*. Let* $T = \mathscr{G}$ *be the sub-probability monad (see [11]). Then the automata of type* T, F *are sub-probabalistic automata.*

3.2 Semantics of Automata

Setting 5. *All conditions from Setting 1 and, in addition, assume that* F *is lifted to a functor* $\overline{F}$ *on* $\mathcal{K}\ell(T)$*, via a distributive law* $\lambda\colon FT \longrightarrow TF$*, and that* $\mathcal{K}\ell(T)$ *has a final* $\overline{F}$*-coalgebra,* $\omega\colon \Omega \longrightarrow F\Omega$*.*

Note that the $\overline{F}$-coalgebras in $\mathcal{K}\ell(T)$ are precisely the automata (of type T, F). The final $\overline{F}$-coalgebra is what we will use in order to abstractly define the semantics for F, T automata:

Definition 6. *Let $\alpha\colon X \longrightarrow \overline{F}X$ in $\mathcal{K}\ell(T)$ be given. We call unique homomorphism into the final coalgebra $[\![-]\!]_\alpha\colon X \longrightarrow \Omega$ the **semantics** of α.*

3.3 Trace and Iterate in General

Before we turn to the study of ε-elimination for these general automata, we present some theory on the assignment $\alpha \mapsto \alpha^\#$. The material is a slight simplification of the work by Hasuo in [7].

Setting 7. *Let $\mathbf{K}$ be a category that has all countable coproducts. Moreover, assume that for each object B from $\mathbf{K}$, there is a final $- + B$-coalgebra,*

$$\xi_B\colon N_B \longrightarrow N_B + B.$$

This setting is equivalent to require that the functor $- + B$ is iteratable [13]. In the sequel we instantiate $\mathbf{K}$ to the Kleisli category of a given monad.

Recall that since ξ_B is final, there is a unique homomorphism from each $- + B$-coalgebra to ξ_B. We call this homomorphism **trace**.

Definition 8. *Let $\beta\colon X \to X + B$ be a morphism in $\mathbf{K}$. The **trace** of β is the unique morphism $\mathrm{tr}_\beta\colon X \to N_B$ such that the following diagram commutes.*

$$\begin{array}{ccc} X & \xrightarrow{\mathrm{tr}_\beta} & N_B \\ {\scriptstyle\beta}\downarrow & & \downarrow{\scriptstyle\xi_B} \\ X + B & \xrightarrow{\mathrm{tr}_\beta + B} & N_B + B \end{array}$$

Setting 9. *Let $\mathbf{K}$ be a category that has all countable coproducts. Let B be an object from $\mathbf{K}$. Denote the initial $- + B$-algebra by $\iota_B\colon \mathbb{N}\cdot B + B \longrightarrow \mathbb{N}\cdot B$. Assume also that $\xi_B := \iota_B^{-1}$ is the final $- + B$-coalgebra. So we have*

$$\xi_B\colon \mathbb{N}\cdot B \longrightarrow \mathbb{N}\cdot B + B.$$

Before we define the **iterate** operator, we need two additional definitions.

Definition 10. *Let $g\colon A \to B$ be a morphism in $\mathbf{K}$. Let $\mathbb{N}\cdot g\colon \mathbb{N}\cdot A \longrightarrow \mathbb{N}\cdot B$ be given by, for all $n \in \mathbb{N}$, $(\mathbb{N}\cdot g)\circ\kappa_n = \kappa_n \circ g$. Equivalently, $\mathbb{N}\cdot g$ is the unique morphism such that*

$$\begin{array}{ccc} \mathbb{N}\cdot A & \xrightarrow{\mathbb{N}\cdot g} & \mathbb{N}\cdot B \\ \downarrow{\scriptstyle\xi_A} & & \downarrow{\scriptstyle\xi_B} \\ \mathbb{N}\cdot A + A & \xrightarrow{\mathbb{N}\cdot g + g} & \mathbb{N}\cdot B + B \end{array} \qquad \textit{commutes.}$$

Definition 11. *Let B be an object of $\mathbf{K}$. The **codiagonal** is the morphism $\nabla_B\colon \mathbb{N}\cdot B \longrightarrow B$ given by $\nabla_B \circ \kappa_n = \mathrm{id}_B$, where $n \in \mathbb{N}$, and $\kappa_n\colon B \longrightarrow \mathbb{N}\cdot B$ is the n-th coprojection.*

Definition 12. *Let X and A be objects from $\mathbf{K}$, and let $\alpha\colon X \longrightarrow X + A$ be a morphism. Then the* ***iterate*** *of α is the morphism $\alpha^{\#}\colon X \longrightarrow A$ given by $\alpha^{\#} := \nabla_A \circ \mathrm{tr}_\alpha$.*

Proposition 13. *Suppose we have a commuting diagram in $\mathbf{K}$ of the form*

$$\begin{array}{ccc} X & \xrightarrow{\quad f \quad} & Y \\ \downarrow{\scriptstyle \alpha} & & \downarrow{\scriptstyle \beta} \\ X + A & \xrightarrow[f+g]{} & Y + B \end{array}$$

where $g\colon A \to B$. Then the following square commutes.

$$\begin{array}{ccc} X & \xrightarrow{\quad f \quad} & Y \\ \downarrow{\scriptstyle \alpha^{\#}} & & \downarrow{\scriptstyle \beta^{\#}} \\ A & \xrightarrow[g]{} & B \end{array}$$

3.4 ε-Elimination in General

First, we define what an abstract automaton with ε-transitions is. (Since our general automata do not explicitly contain an alphabet this is not immediately clear.) Recall that in the case of non-deterministic automata, an automaton with ε-transitions is a map $\alpha\colon X \longrightarrow \wp(\,(A + \{\varepsilon\}) \times X + 1\,)$, and this map gives rise to a second map,

$$\alpha'\colon X \longrightarrow \wp(\,X \;+\; (A \times X + 1)\,).$$

We base our definition on the second map, α', instead of α.

Definition 14. *Let X be an object from $\mathbf{C}$. An* ***ε-automaton*** *of type T, F with states X is a morphism $\alpha\colon X \longrightarrow T(\,X + FX\,)$. In other words, α is an automaton of type T, F_ε, where F_ε is the functor with*

$$F_\varepsilon X \;=\; X + FX.$$

To provide the semantics of ε-automata, we need some assumptions.

Setting 15. *In addition to the assumptions in Setting 5, we assume that $\mathcal{K}\ell(T)$ has a final $\overline{F}_\varepsilon$-coalgebra $\omega_\varepsilon\colon \Omega_\varepsilon \;\overset{\circ}{\longrightarrow}\; \Omega_\varepsilon + F\Omega_\varepsilon$. Here, $\overline{F}_\varepsilon$ is the lifting of F_ε to $\mathcal{K}\ell(T)$, via the distributive law λ_ε given by $(\lambda_\varepsilon)_X \;=\; [T\kappa_\ell, T\kappa_r \circ \lambda_X]$, where X is an object from $\mathbf{C}$. Moreover, let B be an object from $\mathbf{C}$. We denote the initial $- + B$-algebra in $\mathcal{K}\ell(T)$ by $\iota_B\colon \mathbb{N} \cdot B + B \;\overset{\circ}{\longrightarrow}\; \mathbb{N} \cdot B$. Assume that $\xi_B := \iota_B^{-1}$ is the* final *$- + B$-coalgebra in $\mathcal{K}\ell(T)$.*

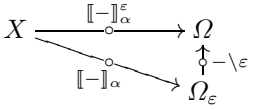
We need a last definition, before providing semantics to ε-automata.

Definition 16. *Let $-\backslash\varepsilon$ be the unique morphism in $\mathbf{C}$ such that*

$$\begin{array}{ccc} \Omega_\varepsilon & \xrightarrow{\quad \cdot\backslash\varepsilon \quad} & \Omega \\ \downarrow{\scriptstyle \omega_\varepsilon^{\#}} & & \downarrow{\scriptstyle \omega} \\ F\Omega_\varepsilon & \xrightarrow[F(\cdot\backslash\varepsilon)]{} & F\Omega \end{array}$$

commutes. That is, $\cdot\backslash\varepsilon$ is the semantics of the automaton $\omega_\varepsilon^{\#}$, $\cdot\backslash\varepsilon \;=\; [\![-]\!]_{\omega_\varepsilon^{\#}}$.

Definition 17. *Let $\alpha\colon X \mathrel{-\!\!\circ\!\!\to} X + FX$ be an ε-automaton of type T, F. The **semantics** of α is the map $[\![-]\!]^{\varepsilon}_{\alpha}\colon X \mathrel{-\!\!\circ\!\!\to} \Omega$ such that*



commutes, where $[\![-]\!]_{\alpha}$ is the semantics of α seen as automaton of type T, F_{ε}.

We can now present one of the main results of this paper, showing that (language) semantics is preserved by the abstract ε-elimination procedure.

Theorem 18 (ε-Elimination). *Let X be from $\mathbf{C}$. Let $\alpha\colon X \mathrel{-\!\!\circ\!\!\to} X+FX$ be an ε-automaton of type T, F. Then the iterate $\alpha^{\#}\colon X \mathrel{-\!\!\circ\!\!\to} FX$ is an automaton of type T, F with the same semantics as α. That is,*

$$[\![-]\!]_{\alpha^{\#}} \;=\; [\![-]\!]^{\varepsilon}_{\alpha}.$$

4 Weighted Automata and the $\mathscr{M}$ Monad

We now briefly return to the case of the weighted automata. Due to space constraints, we leave most details to the reader. Recall that a weighted automaton over a semiring S with states X and alphabet A is a map $\alpha\colon X \longrightarrow \mathscr{M}FX$, where $F = A \times - + 1$. So α is an automaton of type $\mathscr{M}, F$.

Unfortunately, the type $\mathscr{M}, F$ does not fit our general framework for automata (see Setting 15), since the inverse

$$\iota_B^{-1}\colon \mathbb{N}\cdot B \mathrel{-\!\!\circ\!\!\to} \mathbb{N}\cdot B + B$$

of the initial $- + B$-algebra ι_B in $\mathcal{K}\ell(\mathscr{M})$ is not the final $- + B$-coalgebra.

Indeed, this follows from the following example.

Example 19. *Let $B := \{b\}$ and let $\alpha\colon \{*\} \mathrel{-\!\!\circ\!\!\to} \{*\} + B$ be given by*

$$\alpha(*)(\kappa_{\ell}(*)) = \alpha(*)(\kappa_r(b)) = 1.$$

Suppose $\tau\colon \{*\} \mathrel{-\!\!\circ\!\!\to} \mathbb{N}\cdot B$ is a homomorphism from α to ι_B^{-1}. Then $\operatorname{supp}\tau(*)$ is finite by definition of $\mathscr{M}$. However, the reader can verify that $\tau(*)(n, b) = 1$ for all $n \in \mathbb{N}$. So we see that $\operatorname{supp}\tau(*)$ must be infinite as well. No such τ exists. Hence, ι_B^{-1} is not the final $- + B$-coalgebra in $\mathcal{K}\ell(\mathscr{M})$.

In order to study weighted automata in the general framework, we use

$$\mathscr{M}X \;:=\; \{\, \varphi\colon X \to S \mid \operatorname{supp}\varphi \text{ is at most countable} \,\}$$

instead of $\mathscr{M}X$. To turn $\mathscr{M}$ in a monad we need to assume that S is equipped with a notion of countable sums. For more details, see [19].

Note that an automaton of type $\mathscr{M}, F$ represents a weighted automaton that is allowed to have infinitely many (proper) transitions from a given state, while an automaton of type $\mathscr{M}, F$ is a weighted automaton with only finitely many transitions from a given state.

Fortunately, the automata of type $\mathscr{M}, F$ do fit nicely in our framework. That is, Setting 15 applies to them.

Proposition 1. *Given a set B, the inverse $\iota_B^{-1} \colon \mathbb{N} \cdot B \longrightarrow\!\!\!\!\circ\; \mathbb{N} \cdot B + B$ of the initial $- + B$-algebra ι_B in $\mathcal{K}\ell(\mathscr{M})$ is the final $- + B$-coalgebra. Similarly, the inverse $\xi \colon A^* \longrightarrow\!\!\!\!\circ\; A \times A^* + 1$ of the initial $\overline{F}$-algebra is the final $\overline{F}$-coalgebra in $\mathcal{K}\ell(\mathscr{M})$.*

Moreover, given a set A, and $\alpha \colon X \longrightarrow\!\!\!\!\circ\; X + A$ in $\mathcal{K}\ell(\mathscr{M})$, the iterate $\alpha^\#$ of α is given by, for all $q_0 \in X$ and $a \in A$,

$$\alpha^\#(q_0)(a) \;=\; \sum_{n \in \mathbb{N}} \sum_{q_1 \in X} \cdots \sum_{q_{n+1} \in X} \Big(\prod_{i=1}^{n} \alpha(q_i)(q_{i+1}) \Big) \cdot \alpha(a).$$

So we see that the abstract theory gives the expected results: the semantics $[\![-]\!]_\alpha$ of an automaton of type $\mathscr{M}, F$ turns out to be precisely the same as the semantics that we discussed before (see Equation (6)).

4.1 Valid Semirings

There is, however, a catch. The monad $\mathscr{M}$ is only defined over a *σ-semiring*, that is, a semiring S equipped with a summation operation that assigns to each family $(x_i)_{i \in I}$ of elements of S a sum $\sum_{i \in I} x_i$.

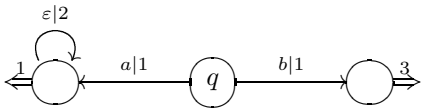
Usually, a semiring S is only equipped with a sum for some families of elements, which are then called *summable*. This idea is formalised in the notion *partial σ-semiring*. An example is the semiring of non-negative reals, $[0, \infty)$, equipped with a sum for all absolutely summable sequences. There are many examples of such partial σ-semirings. In fact, any semiring S is a partial σ-semiring in which only the *finite* families are summable.

It is often possible to extend a partial σ-semiring S to a σ-semiring by adding one element $*$ to S and declaring that the sum of a family of element $(x_i)_{i \in I}$ of $S \cup \{*\}$ is the sum in S when $(x_i)_{i \in I}$ was summable in S and otherwise $*$.

Indeed, the above construction is possible if the partial σ-semiring has the following property: for all $a, b \in S$, $a + b = 0 \implies a = 0$ and $b = 0$. We call such semirings **positive** (using the terminology Gumm introduced for monoids [6]). In fact, any σ-semiring must be positive. So we see that only the positive partial σ-semirings can be extended to a σ-semiring. A typical example of a semiring that is not positive is $\mathbb{R}$.

Let S be a positive partial σ-semiring. Then S can be extended to a σ-semiring $S \cup \{*\}$, and hence the abstract framework for automata is applicable to weighted automata over the semiring $S \cup \{*\}$.

The object $*$ acts as an "undefined" element. Consider the following weighted automaton α with ε-transitions over the semiring $\mathbb{R}$ with alphabet $\{a, b\}$.



Let us compute the semantics of α with Equation (8). We see that $[\![q]\!]^{\varepsilon}_{\alpha}(b) = 3$, but there is a difficulty when computing $[\![q]\!]^{\varepsilon}_{\alpha}(a) = 1 + 2 + 4 + \cdots$. However, if we consider α as a weighted automaton over the semiring $S \cup \{*\}$, then we simply get $[\![q]\!]^{\varepsilon}_{\alpha}(a) = *$, while still $[\![q]\!]^{\varepsilon}_{\alpha}(b) = 3$.

All in all, the abstract framework applies to, and hence given us semantics, ε-elimination, and so on, for *all* weighted automata over *positive* semirings (possibly equipped with a partial summation).

5 Discussion

We have presented a framework where ε-elimination can be thought of in an abstract manner. The framework yields procedures for non-deterministic automata and, notably, for weighted automata. What we presented here can be seen as the beginning of a larger quest to understand multi-step behavior, which is still a challenge coalgebraically. There are several directions we would like to explore further and which we discuss briefly next.

Kleisli versus Eilenberg–Moore. Recovering coalgebraic definitions of language equivalence has been done in two different settings. The one we used in this paper, based on Kleisli categories, and the one presented in [18,17,10], based on Eilenberg-Moore categories and a generalized powerset construction. The definition of iterate is natural in Kleisli and hence we have taken the first approach. We want to explore if it is possible to define similar notions in the Eilenberg-Moore setting and enlarge the examples the framework covers. For instance, the generalized powerset construction works for every weighted automaton, without having to resort to changes in the monad.

Weak Bisimilarity. ε-transitions are in some sense similar to τ-transitions in labelled transition systems (LTS). However, there are some subtleties to be tackled, before fully exploring the present framework to study weak bisimilarity. In particular, consider the example of the processes $\mathtt{a} + \mathtt{b}$ and $\tau.\mathtt{a} + \mathtt{b}$, which are not weakly bisimilar. Naively applying the framework above would erroneously identify them and extra care needs to be taken in order to avoid this. A more detailed account on the applications to weak bisimilarity can be found in [19].

Acknowledgments. We are grateful to Tomasz Brengos, Bart Jacobs, Herman Geuvers, Ichiro Hasuo, Jan Rutten, Pawel Sobocinski and the anonymous referees for useful comments and pointers to the literature. The work of Bram Westerbaan is funded by the NWO project *Practical conduction*. The work of Alexandra

Silva is partially funded by the ERDF through the Programme COMPETE and by the Portuguese Government through FCT - Foundation for Science and Technology, project ref. PTDC/EIA-CCO/122240/2010 and SFRH/BPD/71956/2010.

References

1. Bonchi, F., Bonsangue, M., Boreale, M., Rutten, J., Silva, A.: A coalgebraic perspective on linear weighted automata. Inf. Comput. 211, 77–105 (2012)
2. Bonchi, F., Bonsangue, M., Rutten, J., Silva, A.: Brzozowski's algorithm (co)algebraically. In: Constable, R.L., Silva, A. (eds.) Logic and Program Semantics. LNCS, vol. 7230, pp. 12–23. Springer, Heidelberg (2012)
3. Bonchi, F., Pous, D.: Checking NFA equivalence with bisimulations up to congruence. In: POPL, pp. 457–468. ACM (2013)
4. Brengos, T.: Weak bisimulations for coalgebras over ordered functors. In: Baeten, J.C.M., Ball, T., de Boer, F.S. (eds.) TCS 2012. LNCS, vol. 7604, pp. 87–103. Springer, Heidelberg (2012)
5. Droste, M., Kuich, W., Vogler, W.: Handbook of Weighted Automata. Springer (2009)
6. Peter Gumm, H., Schröder, T.: Monoid-labeled transition systems. ENTCS 44(1), 185–204 (2001)
7. Hasuo, I., Jacobs, B., Sokolova, A.: Generic Forward and Backward Simulations. In: Proceedings of JSSST Annual Meeting (2006) (Partly in Japanese)
8. Hasuo, I., Jacobs, B., Sokolova, A.: Generic Trace Semantics via Coinduction. Logical Methods in Computer Science 3(4) (2007)
9. Jacobs, B.: From coalgebraic to monoidal traces. ENTCS 264(2), 125–140 (2010)
10. Jacobs, B., Silva, A., Sokolova, A.: Trace semantics via determinization. In: Pattinson, D., Schröder, L. (eds.) CMCS 2012. LNCS, vol. 7399, pp. 109–129. Springer, Heidelberg (2012)
11. Kerstan, H., König, B.: Coalgebraic trace semantics for probabilistic transition systems based on measure theory. In: Koutny, M., Ulidowski, I. (eds.) CONCUR 2012. LNCS, vol. 7454, pp. 410–424. Springer, Heidelberg (2012)
12. Lombardy, S., Sakarovitch, J.: The Removal of Weighted ϵ-Transitions. In: Moreira, N., Reis, R. (eds.) CIAA 2012. LNCS, vol. 7381, pp. 345–352. Springer, Heidelberg (2012)
13. Milius, S.: On Iteratable Endofunctors. In: CTCS. ENTCS, vol. 69, pp. 287–304. Elsevier (2002)
14. Mohri, M.: Generic ϵ-removal algorithm for weighted automata. In: Yu, S., Păun, A. (eds.) CIAA 2000. LNCS, vol. 2088, pp. 230–242. Springer, Heidelberg (2001)
15. Rutten, J.: Automata and coinduction (an exercise in coalgebra). In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 194–218. Springer, Heidelberg (1998)
16. Sangiorgi, D.: An introduction to bisimulation and coinduction. Cambridge University Press (2012)
17. Silva, A., Bonchi, F., Bonsangue, M., Rutten, J.: Generalizing the powerset construction, coalgebraically. In: FSTTCS. LIPIcs, vol. 8, pp. 272–283 (2010)
18. Silva, A., Bonchi, F., Bonsangue, M., Rutten, J.: Generalizing determinization from automata to coalgebras. LMCS 9(1) (2013)
19. Silva, A., Westerbaan, B.: A Coalgebraic View of ϵ-Transitions. Extended abstract, with proofs, http://alexandrasilva.org/files/epsilon-extended.pdf
20. Sokolova, A., de Vink, E., Woracek, H.: Coalgebraic weak bisimulation for action-type systems. Sci. Ann. Comp. Sci. 19, 93–144 (2009)

Nets, Relations and Linking Diagrams

Paweł Sobociński

ECS, University of Southampton, UK

Abstract. In recent work, the author and others have studied compositional algebras of Petri nets. Here we consider mathematical aspects of the pure linking algebras that underly them. We characterise composition of nets without places as the composition of spans over appropriate categories of relations, and study the underlying algebraic structures.

Introduction

Linking structures are ubiquitous in Computer Science, Logic and Mathematics. Amongst many examples, we mention Kelly-Laplaza graphs for compact closed categories [13] and proof nets [10]. Linking diagrams[1] underly string diagrams [12, 19] that are used to characterise the arrows of various kinds of free categories. Similar structures have been used by Computer Scientists to develop foundational algebras for composing software components [2, 3]. Theoretical work has led to tool support for reasoning about different kinds of string diagrams [14, 22].

In [4, 5, 20, 21] the author and others have studied compositional algebras of Petri nets. The two main variants, studied in detail in [5], are *C/E nets with boundaries* and *P/T nets with boundaries*. Nets without places are pure algebras of linkings; we show in this paper that they are, respectively, the arrows of two categories $\mathsf{Sp}(\mathbf{Rel}_f^c)$ and $\mathsf{Spr}(\mathbf{Rel}_f^{\mathcal{M}})$[2]. Recently, string diagrams and closely related algebraic structures have also been used to reason about quantum computation [1, 7, 18].

Both categories are generated from a set of basic components, which are the building blocks of *two different* monoid-comonoid structures on the underlying categories. The two structures arise, roughly, from the elementary setting of cospans and spans of finite sets.

In an effort to capture several different kinds of linking algebras, Hughes [11] introduced the category Link of spans over $\mathbf{iRel}$ the category of injective relations, which has pullbacks. Pullbacks are obtained by considering paths, called *minimal synchronisations*, in the corresponding linking diagrams. Similar ideas are used here in order to construct pullbacks in $\mathbf{Rel}_f^c$, the category of *relations*

[1] We use this terminology loosely to mean "string diagrams without boxes."

[2] The notation $\mathsf{Sp}(-)$ means "not quite the category of spans," as the objects are the natural numbers, instead of arbitrary sets. Similarly $\mathsf{Spr}(-)$ is "not quite the category of relational spans," where relational means that the two legs are jointly mono. Both categories are PROPs [16, 17].

R. Heckel and S. Milius (Eds.): CALCO 2013, LNCS 8089, pp. 282–298, 2013.

with contention and weak pullbacks in $\mathbf{Rel}_f^{\mathcal{M}}$, the category of multirelations. In this paper we study only finite linkings but the category of spans of relations with contention is more expressive than the category of spans of injective relations: the finite counterpart of Hughes' category Link embeds into $\mathsf{Sp}(\mathbf{Rel}_f^c)$.

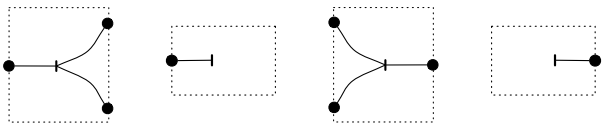
Structure of the Paper. In §1 we introduce the two monoid-comonoid structures that arise from considering cospans and spans of finite sets. In §2 we introduce sets and relations with contention, and show that the category of the latter has pullbacks. This allows us, in §3 to consider the category $\mathsf{Sp}(\mathbf{Rel}_f^c)$, a universe where both the monoid-comonoid structures can be considered. In §4 we discuss multirelations and construct weak pullbacks, which we then use in §5 to consider another universe where both the monoid-comonoid structures exist and interact.

Notational Conventions. Relations from X to Y are identified with functions $X \to 2^Y$. For $k \in \mathbb{N}$ we abuse notation and denote the kth finite ordinal $\{0, 1 \ldots, k-1\}$ with k. For sets X, Y, $X+Y \stackrel{\text{def}}{=} \{\, (x,0) \,|\, x \in X \,\} \cup \{\, (y,1) \,|\, y \in Y \,\}$. Functions are labelled with ! when there is a unique function with that particular domain and codomain, $tw : 2 \to 2$ is the function $tw(0) = 1$ and $tw(1) = 0$. Given a function $f : X \to Y$, $[f] \subseteq X \times Y$ is its graph: $[f] \stackrel{\text{def}}{=} \{\, (x, fx) \,|\, x \in X \,\}$. Given a relation $R \subseteq X \times Y$, $R^{\mathsf{op}} \subseteq Y \times X$ is the opposite relation.

1 Components of Linking Diagrams

Let $\mathsf{Csp}(\mathbf{Set}_f)$ be the category[3] with objects the natural numbers, and arrows isomorphism classes of cospans $k \to x \leftarrow l$, where k and l are considered as finite ordinals. Composition is obtained via pushout in $\mathbf{Set}_f$, associativity follows from the universal property. Given $k_1 \to m_1 \leftarrow l_l$ and $k_2 \to m_2 \leftarrow l_2$, the tensor product is $k_1 + k_2 \to m_1 + m_2 \leftarrow l_1 + l_2$.

The following diagrams represent certain arrows in $\mathsf{Csp}(\mathbf{Set}_f)$. They have



$(\Delta \perp \nabla \top)$

representatives $1 \xrightarrow{\text{id}} 1 \xleftarrow{!} 2$, $1 \xrightarrow{\text{id}} 1 \xleftarrow{!} 0$, $2 \xrightarrow{!} 1 \xleftarrow{\text{id}} 1$ and $0 \xrightarrow{!} 1 \xleftarrow{\text{id}} 1$.

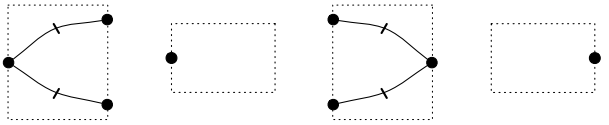
Our graphical notation calls for further explanation: within the diagrams, each *link*–an undirected multiedge–represents an element of the carrier set, its connections to *boundary ports* (elements of the ordinals on the boundary) are determined in $\mathsf{Csp}(\mathbf{Set}_f)$ by the functions from the ordinals that represent the boundaries. Each link has a small perpendicular mark; this is used to distinguish between different links within diagrams.

[3] Not quite the category of cospans. Again, this is a PROP.

The definition of $\mathsf{Csp}(\mathbf{Set}_f)$ enforces some structural restrictions on links. Indeed, each boundary port must be connected to exactly one link; ie no two links can be connected to the same boundary port. Any link, however, can be connected to several ports on each boundary.

Now consider $\mathsf{Sp}(\mathbf{Set}_f)$, the category with objects the natural numbers, and arrows isomorphism classes of spans $k \leftarrow x \rightarrow l$, where k and l are considered as finite ordinals. Composition is obtained via pullback in $\mathbf{Set}_f$, and associativity is again guaranteed by a universal property, this time of pullbacks. Again, $+$ gives a tensor product.

The following diagrams represent certain arrows in $\mathsf{Sp}(\mathbf{Set}_f)$. They have

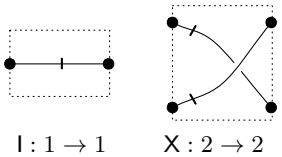


$(\Lambda\downarrow \mathsf{V}\uparrow)$

representatives $1 \xleftarrow{!} 2 \xrightarrow{\mathrm{id}} 2$, $1 \xleftarrow{!} 0 \xrightarrow{\mathrm{id}} 0$, $2 \xleftarrow{\mathrm{id}} 2 \xrightarrow{!} 1$ and $0 \xleftarrow{\mathrm{id}} 0 \xrightarrow{!} 1$.

In the diagrams, the links again represent elements of the carrier set but connections to boundary ports are now given by the functions *from* the carrier *to* the boundaries. Due to the definition of $\mathsf{Sp}(\mathbf{Set}_f)$, there are again structural restrictions: each link is connected to exactly one port on each boundary. Any port, however, can be connected to many links.

The following diagrams represent certain arrows in $\mathsf{Csp}(\mathbf{Set}_f)$ and $\mathsf{Sp}(\mathbf{Set}_f)$. As (isomorphism classes of) cospans they are $1 \rightarrow 1 \leftarrow 1$, $2 \xrightarrow{tw} 2 \leftarrow 2$, as spans



$(\mathsf{I}\,\mathsf{X})$

they are $1 \leftarrow 1 \rightarrow 1$, $2 \leftarrow 2 \xrightarrow{tw} 2$.

1.1 The Algebra of $\mathsf{Csp}(\mathbf{Set}_f)$

In Fig. 1 we give some of the equations satisfied by the algebra generated from the components $(\Delta\,\bot\,\nabla\,\top)$ and $(\mathsf{I}\,\mathsf{X})$ in $\mathsf{Csp}(\mathbf{Set}_f)$: $(\Delta\mathrm{UC})$ and $(\Delta\mathrm{A})$ show that Δ is the comultiplication of a cocommutative comonoid. The symmetric equations hold for ∇, meaning that it is part of a commutative monoid structure. The Frobenius axioms (F) [6,15] hold, and the algebra is separable (S). In fact $\mathsf{Csp}(\mathbf{Set}_f)$ is the free PROP on $(\Delta\,\bot\,\nabla\,\top)$ satisfying such axioms, where (F), (S) can be understood as witnessing a distributive law of PROPs; see [16] for the details. In (CC) we indicate how the (self dual) compact closed structure of $\mathsf{Csp}(\mathbf{Set}_f)$ arises.

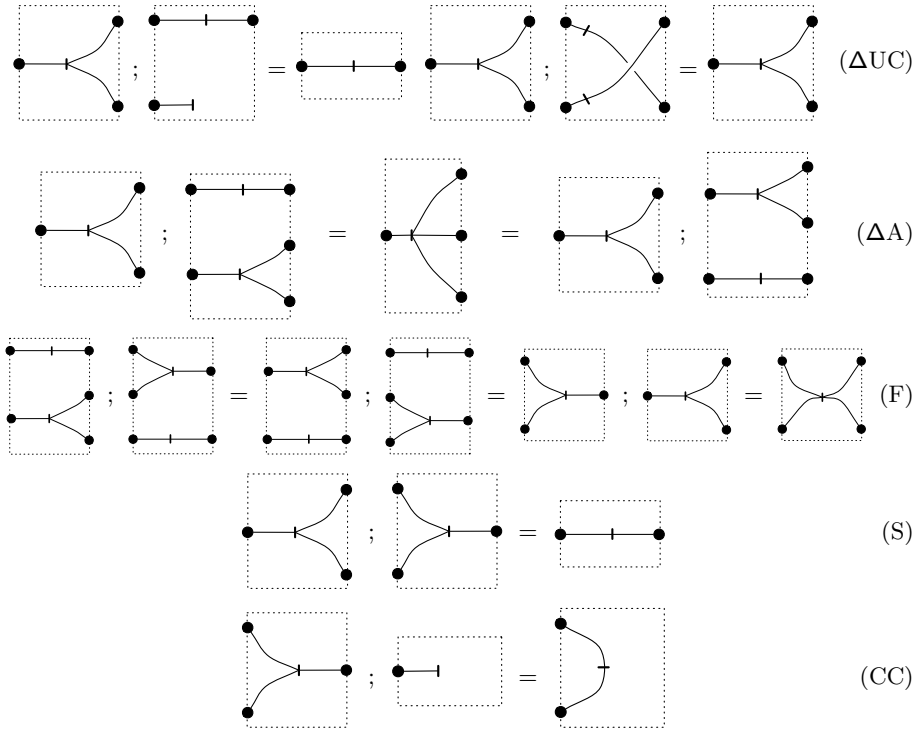


Fig. 1. Equations in $\mathsf{Csp}(\mathbf{Set}_f)$

1.2 The Algebra of $\mathsf{Sp}(\mathbf{Set}_f)$

In Fig. 2 we exhibit some equations satisfied by the components $(\Lambda \downarrow \mathsf{V} \uparrow)$ and $(\mathsf{I}\,\mathsf{X})$ in $\mathsf{Sp}(\mathbf{Set}_f)$: $(\Lambda\mathrm{UC})$ and $(\Lambda\mathrm{A})$ show that Λ is the multiplication of a cocommutative comonoid, similarly the symmetric equations, which we do not illustrate, show that that V is a commutative monoid. Differently from Fig. 1, here the Frobenius equations do not hold; but rather the equations of commutative and cocommutative bialgebras: in (B), $(\mathsf{V}\downarrow)$ and $(\Lambda\mathsf{V})$ we show how the monoid and comonoid structures interact in $\mathsf{Sp}(\mathbf{Set}_f)$. In fact, $\mathsf{Sp}(\mathbf{Set}_f)$ is the free PROP on $(\Lambda \downarrow \mathsf{V} \uparrow)$ satisfying the equations of commutative and cocommutative bialgebras, and the bialgebra axiom can be understood as a distributive law of PROPs, see [16].

1.3 Bringing It All Together

Note that none of the diagrams in $(\Delta \perp \nabla \top)$ represent valid spans: for instance the link in Δ connects to two different ports on its right boundary, and the link in $\perp$ does not connect to any port on its right boundary. Similarly, none of $(\Lambda \downarrow \mathsf{V} \uparrow)$ represent valid cospans. Thus, for mundane "expressivity" reasons, $(\Delta \perp \nabla \top)$ are not arrows of $\mathsf{Sp}(\mathbf{Set}_f)$, and vice-versa, $(\Lambda \downarrow \mathsf{V} \uparrow)$ are not arrows

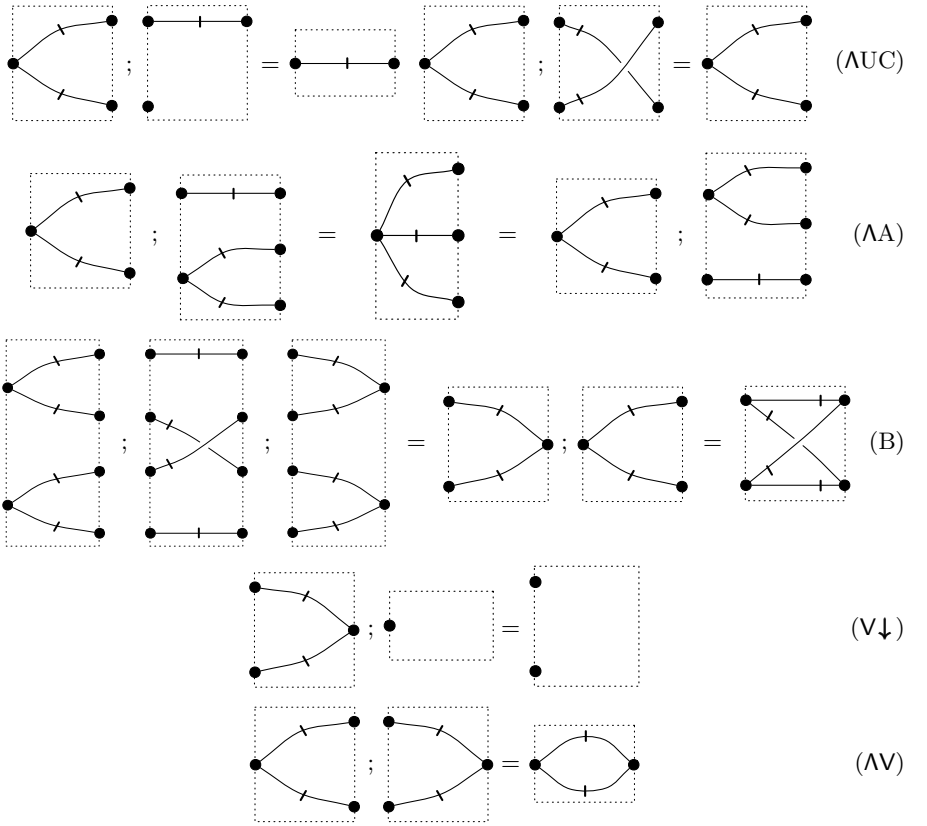


Fig. 2. Equations in $\mathsf{Sp}(\mathbf{Set}_f)$

of $\mathsf{Csp}(\mathbf{Set}_f)$. The remit of this paper is to study how these two commutative monoid-comonoid structures interact together in universes that are expressive enough to accommodate them.

For example, instead of studying cospans and spans of *functions*, one could consider spans (or cospans) of *relations*. Indeed, it is not difficult to check that all of the components $(\Delta \perp \nabla \top)$, $(\wedge \downarrow \vee \uparrow)$ and $(\mathsf{I}\mathsf{X})$ are spans of relations of finite sets. The problem, of course, is that $\mathbf{Rel}_f$, the category of finite sets and relations, does not have pullbacks nor pushouts: it is thus not clear how to define the composition of such linking diagrams.

In the following sections we study two different universes that are expressive enough to contain $(\Delta \perp \nabla \top)$, $(\wedge \downarrow \vee \uparrow)$ and $(\mathsf{I}\mathsf{X})$ and the intriguing, different ways in which the two monoid/comonoid structures interact in the universes. They arose through the study of compositional algebras of Petri nets with boundaries [4, 5, 20, 21].

2 Sets with Contention

In this section we introduce *sets with contention*, over which one can define a category of relations that has pullbacks, and is expressive enough to accommodate the components ($\Delta \perp \nabla \top$), ($\wedge \downarrow \vee \uparrow$) and ($\mathsf{I} \mathsf{X}$).

A *set with contention*, or c-set, is a pair $(X, \bowtie_X)$, where X is a set and $\bowtie_X \subseteq X \times X$ is a reflexive ($\forall x \in X.\ (x \bowtie_X x)$) and symmetric ($\forall x, y \in X.\ (x \bowtie_X y) \Rightarrow (y \bowtie_X x)$) relation called *contention*.[4] The complement relation $\between_X$ is called *independence*. To describe a c-set it is thus of course enough to specify either contention or independence. Sets with contention of the form (X, δ_X), where $\delta_X = \{ (x, x) \mid x \in X \}$, are said to be *discrete*. We will normally write simply X for the pair $(X, \bowtie_X)$.

A *morphism of c-sets* $f : X \to Y$ is a function $f : X \to Y$ such that:

$$\forall x, x' \in X.\ f(x) \bowtie_Y f(x') \Rightarrow x \bowtie_X x' \tag{1}$$

(or equivalently $\forall x, x' \in X,\ x \between_X x'$ implies $fx \between_Y fx'$.) The category of finite c-sets and their morphisms is denoted $\mathbf{Set}^c_f$.

Given c-sets X_0 and X_1, $X_0 + X_1$ is the c-set with $X_0 + X_1$ as its underlying set and $(x, i) \bowtie_{X_0+X_1} (y, j)$ iff $i = j$ and $x \bowtie_{X_i} y$. This is the categorical coproduct in $\mathbf{Set}^c_f$.

Given a c-set X, $U \subseteq X$ is said to be *independent* when

$$\forall u, u' \in U.\ u \bowtie_X u' \ \Rightarrow \ u = u'. \tag{2}$$

Let $\mathcal{P}_c X$ denote the set of independent subsets of X. There is functor $\mathcal{P}_c : \mathbf{Set}^c_f \to \mathbf{Set}^c_f$ that takes a c-set X to the set of independent subsets $\mathcal{P}_c X$, with contention between subsets defined:

$$U \bowtie_{\mathcal{P}_c X} V \text{ iff } \exists u \in U, v \in V, u \bowtie_X v.$$

Note that independent subsets are closed under intersection and set difference: indeed, if $U' \subseteq U$ and U is independent then also U' is independent. They are not, in general, closed under union.

If $f : X \to Y$ is a morphism, then letting

$$\mathcal{P}_c f(U) \stackrel{\text{def}}{=} \{ fu \mid u \in U \}$$

defines a morphism $\mathcal{P}_c f : \mathcal{P}_c X \to \mathcal{P}_c Y$ in $\mathbf{Set}^c_f$, since:

(i) given U, for all $u, u' \in U$ if $f(u) \bowtie_Y f(u')$ means that $u \bowtie_X u'$. But U is independent, and thus $u = u'$ and $f(u) = f(u')$, thus $\mathcal{P}_c f(U)$ is an independent subset of Y (recall (2)).

(ii) if $\mathcal{P}_c f(U) \bowtie_Y \mathcal{P}_c f(V)$ then there exists $u \in U$, $v \in V$, such that $f(u) \bowtie_Y f(v)$, so $u \bowtie_X v$ and thus $U \bowtie_X V$, thus $\mathcal{P}_c f$ satisfies (1).

[4] A useful intuition is that links carry signals. When two links are in contention they cannot transmit concurrently. With this intuition ($\Delta \perp \nabla \top$) are copy and forget operations, while ($\wedge \downarrow \vee \uparrow$) are non-deterministic switches and "failure."

2.1 Relations with Contention

There are morphisms $\mu_X : \mathcal{P}_c^2 X \to \mathcal{P}_c X$ with $\{U_i\} \mapsto \bigcup_i U_i$ and a morphism $\eta_X : X \to \mathcal{P}_c X$. It is not difficult to check that they are natural transformations that satisfy the monad axioms.

Let $\mathbf{Rel}_f^c \stackrel{\text{def}}{=} \mathsf{Kl}(\mathcal{P}_c)$ of relations with contention, or c-relations, be the Kleisli category with objects finite c-sets. Arrows from X to Y are morphisms $f : X \to \mathcal{P}_c Y$ in $\mathbf{Set}_f^c$, which we will sometimes denote $f : X \nrightarrow Y$. Given a morphism $f : X \to \mathcal{P}_c Y$ in $\mathbf{Set}_f^c$ (or equivalently, a morphism of $\mathbf{Rel}_f^c$), $f^\# : \mathcal{P}_c X \to \mathcal{P}_c Y$ is the morphism $f^\# U \stackrel{\text{def}}{=} \bigcup_{u \in U} fu$.

The following lemma is useful when calculating in $\mathbf{Rel}_f^c$. It does not hold in $\mathbf{Rel}_f$, the category of ordinary finite sets and relations.

Lemma 21. *Suppose $f : X \to \mathcal{P}_c Y$ in $\mathbf{Set}_f^c$. Then, given $U, U' \in \mathcal{P}_c X$ with $U \subseteq U'$, $f^\#(U' \backslash U) = f^\#(U') \backslash f^\#(U)$. Also, given $U, V, V' \in \mathcal{P}_c X$, with $V \subseteq U$, $V' \subseteq U$, we have $f^\#(V \cap V') = f^\#(V) \cap f^\#(V')$.*

Proof. Since U' is independent, $\{fu\}_{u \in U'}$ is a family of disjoint, independent subsets of Y. Similarly $V \cup V'$ is independent, since they are both subsets of an independent set; and $\{fu\}_{u \in V \cup V'}$ is a family of disjoint, independent subsets of Y. Disjointness implies the desired conclusions. □

2.2 Pullbacks in $\mathbf{Rel}_f^c$

Suppose that $f : A \nrightarrow X$ and $g : B \nrightarrow X$ in $\mathbf{Rel}_f^c$. Given $U \in \mathcal{P}_c A$, $V \in \mathcal{P}_c B$, say that (U, V) is a (f,g)-*synchronisation*[5] if $f^\# U = g^\# V$. We will typically infer f and g from the context and write '$\langle U \curlyvee V \rangle$' as shorthand for 'a synchronisation (U, V)'. Synchronisations inherit an ordering from the subset ordering, pointwise:

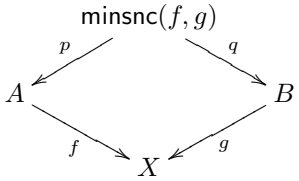
$$\langle U \curlyvee V \rangle \subseteq \langle U' \curlyvee V' \rangle \stackrel{\text{def}}{=} U \subseteq U' \wedge V \subseteq V'.$$

The *trivial* synchronisation is $\langle \varnothing \curlyvee \varnothing \rangle$. A synchronisation $\langle U \curlyvee V \rangle$ is said to be *minimal* when it is not trivial and for all $\langle U' \curlyvee V' \rangle$ such that $\langle U' \curlyvee V' \rangle \subseteq \langle U \curlyvee V \rangle$, either $\langle U' \curlyvee V' \rangle$ is trivial or equal to $\langle U \curlyvee V \rangle$.

Let $\mathsf{minsnc}(f, g)$ be the set of minimal synchronisations of f and g. We can define contention on this set by letting

$$\langle U \curlyvee V \rangle \bowtie_{\mathsf{minsnc}(f,g)} \langle U' \curlyvee V' \rangle \quad \stackrel{\text{def}}{=} \quad U \bowtie_{\mathcal{P}A} U' \ \vee \ V \bowtie_{\mathcal{P}B} V'.$$

It follows that have the following commutative diagram in $\mathbf{Rel}_f^c$



(3)

[5] Hughes [11] uses the term synchronisation in a similar context, and the term has been used in [4, 5, 20] to compose Petri nets with boundaries.

where $p\langle U \curlyvee V\rangle = U$ and $q\langle U \curlyvee V\rangle = V$. The following observations will lead us to conclude in Lemma 24 that the diagram is a pullback in $\mathbf{Rel}_f^c$.

Synchronisations are not in general closed under (pointwise) union, because if $\langle U \curlyvee V\rangle$ and $\langle U' \curlyvee V'\rangle$ then in general it is not true that $U \cup U' \in \mathcal{P}_c A$ and $V \cup V' \in \mathcal{P}_c B$. It is true, however, that the union of any set of minimal synchronisations contained in any synchronisation is again a synchronisation: this is guaranteed by the following.

Lemma 22. *Suppose that $\langle U' \curlyvee V'\rangle \neq \langle U'' \curlyvee V''\rangle$ are minimal synchronisations contained in $\langle U \curlyvee V\rangle$. Then $U' \cap U'' = \varnothing$ and $V' \cap V'' = \varnothing$.*

Proof. By the conclusion of Lemma 21, $f^{\#}(U \cap U') = f^{\#}U \cap f^{\#}U' = g^{\#}V \cap g^{\#}V' = g^{\#}(V \cap V')$, so $\langle U' \cap U'' \curlyvee V' \cap V''\rangle$; by minimality of $\langle U' \curlyvee V'\rangle$ and $\langle U'' \curlyvee V''\rangle$ it follows that $\langle U' \cap U'' \curlyvee V' \cap V''\rangle$ is trivial. □

Lemma 23. *$\langle U \curlyvee V\rangle$ is the union of min. synchronisations it contains.*

Proof. Let $\{\langle U_i \curlyvee V_i\rangle\}_{i\in I}$ be the set of minimal synchronisations contained in $\langle U \curlyvee V\rangle$ and $\langle U' \curlyvee V'\rangle \stackrel{\text{def}}{=} \bigcup_i \{\langle U_i \curlyvee V_i\rangle\}$, then clearly we have $\langle U' \curlyvee V'\rangle \subseteq \langle U \curlyvee V\rangle$. Let $U'' = U \backslash U'$ and $V'' = V \backslash V'$. Now, using the conclusion of Lemma 21, $\langle U'' \curlyvee V''\rangle$, and thus it is either null or it contains a minimal synchronisation. But $\{\langle U_i \curlyvee V_i\rangle\}_{i\in I}$ contains all minimal synchronisations in $\langle U \curlyvee V\rangle$; thus $U'' = V'' = \varnothing$ and we are finished. □

Lemma 24. *The square* (3) *is a pullback diagram in $\mathbf{Rel}_f^c$.*

Proof. Suppose Z is a c-set and $\alpha : Z \to A$, $\beta : Z \to B$ are morphisms in $\mathbf{Rel}_f^c$ such that $f\alpha = g\beta$. In particular, this means that for all $z \in Z$, we have $\langle \alpha z \curlyvee \beta z\rangle$. Define $h : Z \to \mathsf{minsnc}(f, g)$ by letting hz be the family of minimal synchronisations contained in $\langle \alpha z \curlyvee \beta z\rangle$. This is a independent set, due to Lemma 22, and the fact that αz and βz are independent. Then, by the conclusion of Lemma 23, $ph = \alpha$ and $qh = \beta$.

If another h' satisfies $ph' = \alpha$ and $qh' = \beta$ then there exists a family of minimal synchronisations $h'z = \{\langle U_i \curlyvee V_i\rangle\}_{i\in I}$ such that $\bigcup_i U_i = \alpha z$ and $\bigcup_i V_i = \beta z$. By the conclusion of Lemma 22 this family must be hz. □

3 The Algebra of $\mathbf{Sp}(\mathbf{Rel}_f^c)$

In this section we consider a category with enough structure for all of $(\Delta \bot \nabla \top)$, $(\wedge \downarrow \vee \uparrow)$ and $(\mathsf{I} \mathsf{X})$. It has been considered as part of a compositional algebra of C/E (1 bounded) nets [5, 20]—indeed, it is the category of C/E nets with boundaries, without net places, up to isomorphism.

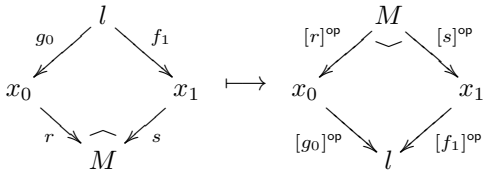
Consider the category $\mathsf{Sp}(\mathbf{Rel}_f^c)$, that has objects the natural numbers and arrows $k \to l$ isomorphism classes of spans $k \stackrel{f}{\leftarrow\!\!\!\cdot} (X, \bowtie_X) \stackrel{g}{\cdot\!\!\!\rightarrow} l$ in $\mathbf{Rel}_f^c$, where k and l are considered as discrete c-sets. Composition is via pullback in $\mathbf{Rel}_f^c$; associativity follows from the universal property. There is a tensor product, given by $+$.

$\mathsf{Sp}(\mathbf{Rel}_f^c)$ has enough structure for $(\Delta \perp \nabla\, \mathsf{T})$, $(\wedge \downarrow \vee \uparrow)$ and $(\mathsf{I}\mathsf{X})$. Indeed, $(\Delta \perp \nabla\, \mathsf{T})$ are, respectively, spans $1 \xleftarrow{[\mathrm{id}]} 1 \xrightarrow{[!]^{\mathsf{op}}} 2$, $1 \xleftarrow{[\mathrm{id}]} 1 \xrightarrow{[!]^{\mathsf{op}}} 0$, $2 \xleftarrow{[!]^{\mathsf{op}}} 1 \xrightarrow{[\mathrm{id}]} 1$ and $0 \xleftarrow{[!]^{\mathsf{op}}} 1 \xrightarrow{[\mathrm{id}]} 1$. Similarly, $(\mathsf{I}\mathsf{X})$ are spans $1 \xleftarrow{[\mathrm{id}]} 1 \xrightarrow{[\mathrm{id}]}$ and $2 \xleftarrow{[\mathrm{id}]} 2 \xrightarrow{[tw]} 2$. Indeed, $\mathsf{Csp}(\mathbf{Set}_f)$ embeds into $\mathsf{Sp}(\mathbf{Rel}_f^c)$.

Theorem 1. *There is a faithful functor $E : \mathsf{Csp}(\mathbf{Set}_f) \to \mathsf{Sp}(\mathbf{Rel}_f^c)$ that is identity-on-objects.*

Proof. A cospan $k \xrightarrow{f} x \xleftarrow{g} l$ is taken to $k \xleftarrow{[f]^{\mathsf{op}}} x \xrightarrow{[g]^{\mathsf{op}}} l$, where k, x and l are discrete c-sets, and $[f]^{\mathsf{op}}$, $[g]^{\mathsf{op}}$ are the opposites of graphs of, respectively, f and g. As arrows in $\mathsf{Kl}(\mathcal{P}_c)$, $[f]^{\mathsf{op}}u = f^{-1}u$ and $[g]^{\mathsf{op}}u = g^{-1}u$ for any $u \in x$. Identities are clearly preserved.

We must show that composition is preserved; it suffices to show that, given $g_0 : l \to x_0$ and $f_1 : l \to x_1$, a pushout diagram of g_0, f_1 in $\mathbf{Set}_f$ is taken to a pullback diagram in $\mathbf{Rel}_f^c$, as illustrated below.



(4)

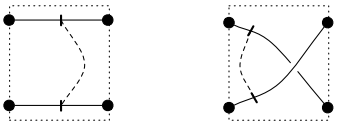
If $M = 0$ then also $x_0 = x_1 = l = 0$ and all arrows are id_0. Otherwise, by an inductive argument it suffices to consider the case $M = 1$. In that case, if $x_0 = 0$ then $x_1 = 1$ and $l = 0$. Then $\mathsf{minsnc}([g_0]^{\mathsf{op}}, [f_1]^{\mathsf{op}}) = \{\langle \varnothing \curlyvee 1 \rangle\}$ and we are done. The case $x_1 = 0$ is symmetric. If both $x_0, x_1 \neq 0$ then clearly $\langle x_0 \curlyvee x_1 \rangle$. In fact, it is the only non-trivial synchronisation (and thus minimal). To see this, notice that g_0 and f_1 are surjective and therefore, if $\langle U_1 \curlyvee V_1 \rangle$ and $\langle U_2 \curlyvee V_2 \rangle$ are two different non-trivial synchronisations then $l_1 \stackrel{\mathrm{def}}{=} g_0^{-1}U_1 = f_1^{-1}V_1 \neq \varnothing$ and $l_2 \stackrel{\mathrm{def}}{=} g_0^{-1}U_2 = f_1^{-1}V_2 \neq \varnothing$, but $l_1 \cap l_2 = \varnothing$. This means that $l = l_1 + l_2 + l_3$, for some l_3, and the whole left hand side of (4) decomposes into a sum, contradicting the assumption that $M = 1$.

The inductive argument relies on sums being compatible with pullbacks in $\mathbf{Rel}_f^c$. This follows from the construction: minimal synchronisations of $x_0 + x_0' \xrightarrow{[g_0 + g_0']^{\mathsf{op}}} l + l' \xleftarrow{[f_0 + f_1']^{\mathsf{op}}} x_1 + x_1'$ arise either as a minimal synchronisations of $[g_0]^{\mathsf{op}}$ and $[f_1]^{\mathsf{op}}$, or those of $[g_0']^{\mathsf{op}}$ and $[f_1']^{\mathsf{op}}$. □

As a consequence, the equations for $(\Delta \perp \nabla\, \mathsf{T})$ —presented in $(\Delta\mathrm{UC})$, $(\Delta\mathrm{A})$, (F), (S) and (CC)— also hold in $\mathsf{Sp}(\mathbf{Rel}_f^c)$.

Also $(\wedge \downarrow \vee \uparrow)$ are spans of c-relations: $1 \xleftarrow{[!]} (2, 2 \times 2) \xrightarrow{[\mathrm{id}]} 2$, $1 \xleftarrow{[!]} 0 \xrightarrow{[\mathrm{id}]} 0$, $2 \xleftarrow{[\mathrm{id}]} (2, 2 \times 2) \xrightarrow{[!]} 1$ and $0 \xleftarrow{[\mathrm{id}]} 0 \xrightarrow{[!]} 1$; notice that contention is used to "encode" $(\wedge \downarrow \vee \uparrow)$. This is necessary because the two elements of 2 must be in contention in order for $! : 2 \to 1$ to be a c-morphism.

Remark 1. When considering, for instance Λ of $(\Lambda\downarrow\vee\uparrow)$ we are in a situation where two links connect to the same point on the boundary. Since any element is in contention with itself, this means that the two links must be in contention. Thus, in this example, contention between the two links is implied and we will not alter our graphical notation. We will, however, need a way to represent contention graphically when it is not implied "structurally," and we will do this by connecting the links with dotted lines. For instance, the two diagrams below represent the spans $2 \xleftarrow{[\mathrm{id}]} (2, 2\times 2) \xrightarrow{[\mathrm{id}]} 2$ and $2 \xleftarrow{[\mathrm{id}]} (2, 2\times 2) \xrightarrow{[tw]} 2$.

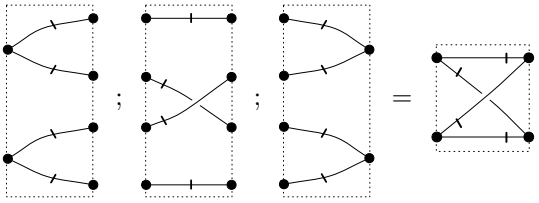


(5)

Remark 2. There is also an "embedding" $F : \mathsf{Sp}(\mathbf{Set}_f) \to \mathsf{Sp}(\mathbf{Rel}_f^c)$. A span $k \xleftarrow{f} x \xrightarrow{g} l$ is sent to the span $k \xleftarrow{[f]} (x, x\times x) \xrightarrow{[g]} l$, with the carrier set having all elements in contention. It is not difficult to check that composition is preserved, but the mapping fails to be a functor because identities are not preserved. For instance, the identity on 2 is mapped to the left diagram of (5), which is not the identity on 2 in $\mathsf{Sp}(\mathbf{Rel}_f^c)$.

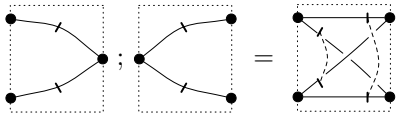
The finite fragment of Hughes' category Link of spans of *injective* relations [11] lies between $\mathsf{Csp}(\mathbf{Set}_f)$ and $\mathsf{Sp}(\mathbf{Rel}_f^c)$. Indeed, spans of injective relations are expressive enough to consider all the structure of $(\Delta\perp\nabla\top)$, $(\mathsf{I}\mathsf{X})$ and the units $\downarrow$, $\uparrow$ of $(\Lambda\downarrow\vee\uparrow)$; but not the comultiplication and multiplication Λ, $\vee$ — these are not injective relations. Link embeds into $\mathsf{Sp}(\mathbf{Rel}_f^c)$, thus all the equations that hold in the former hold also in the latter. We omit the details here.

Equations $(\Lambda\mathrm{UC})$, $(\Lambda\mathrm{A})$, $(\vee\downarrow)$ and $(\Lambda\vee)$ hold in in $\mathsf{Sp}(\mathbf{Rel}_f^c)$. Equation (B) does not hold: while we have



(B^c)

we have

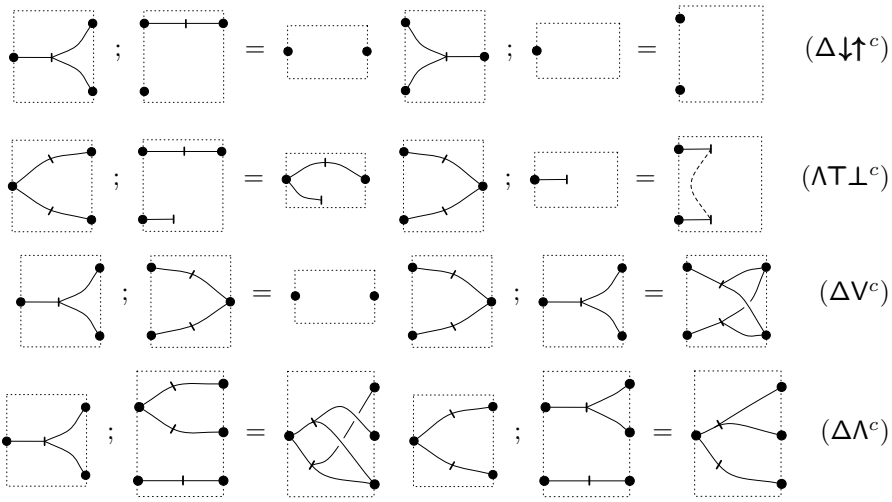


$(\vee\Lambda^c)$

and the right-hand sides are not equal as arrows of $\mathsf{Sp}(\mathbf{Rel}_f^c)$.

In $(\Delta\downarrow\uparrow^c)$, $(\Lambda\top\perp^c)$, $(\Delta\vee^c)$ and $(\Delta\Lambda^c)$ below we show how $(\Delta\perp\nabla\top)$ and $(\Lambda\downarrow\vee\uparrow)$ interact together in $\mathsf{Sp}(\mathbf{Rel}_f^c)$. We comment on two of the more interesting equations that the interactions suggest: the right hand side of $(\Delta\vee^c)$

implies $\mathsf{V} \;;\; \Delta = (\Delta \otimes \Delta) \;;\; (\mathsf{I} \otimes \mathsf{X} \otimes \mathsf{I}) \;;\; (\mathsf{V} \otimes \mathsf{V})$, an "asymmetric" commutative/cocommutative bialgebra structure. The left hand side of $(\Delta\Lambda^c)$ implies $\Delta \;;\; (\Lambda \otimes \mathsf{I}) = \Lambda \;;\; (\Delta \otimes \Delta) \;;\; (\mathsf{I} \otimes \mathsf{X} \otimes \mathsf{I}) \;;\; (\mathsf{I} \otimes \mathsf{I} \otimes \mathsf{V})$.



All linking diagrams in $\mathsf{Sp}(\mathbf{Rel}_f^c)$ can be obtained from the basic set of components $(\Delta \bot \nabla \top)$, $(\Lambda \downarrow \mathsf{V} \uparrow)$ and $(\mathsf{I}\mathsf{X})$, combined using the operations of composition and tensor.

Theorem 2. *Every arrow in* $\mathsf{Sp}(\mathbf{Rel}_f^c)$ *decomposes into an expression consisting only of* Δ, $\bot$, ∇, $\top$, Λ, $\downarrow$, V, $\uparrow$, I, X, *composed with* ; *and* $\otimes$.

Proof. Omitted.

4 Multisets and Multirelations

We have seen that $\mathsf{Sp}(\mathbf{Rel}_f^c)$ is a setting in which one can study the algebra of $(\Delta \bot \nabla \top)$, $(\Lambda \downarrow \mathsf{V} \uparrow)$ and $(\mathsf{I}\mathsf{X})$. Here we develop a second, different setting, that arises from a compositional algebra of P/T nets [5].

Given a set X, let $\mathcal{M}_X$ denote the set of finite maps $\mathcal{U} : X \to \mathbb{N}$, ie where $dom(\mathcal{U})$ is a finite set. We call elements of $\mathcal{M}_X$ *multisets*. We will sometimes abuse set notation to when talking about multisets; any ordinary set $U \subseteq X$ can be considered as a multiset in the obvious way:

$$Ux = \begin{cases} 1 & \text{if } x \in U \\ 0 & \text{otherwise.} \end{cases}$$

Given $\mathcal{U}, \mathcal{V} \in \mathcal{M}_X$, $\mathcal{U}+\mathcal{V}$ is the multiset $(\mathcal{U}+V)(x) = \mathcal{U}x + \mathcal{V}x$. We say $\mathcal{U} \geq \mathcal{V}$ if $\forall x.\, \mathcal{U}x - \mathcal{V}x \in \mathbb{N}$. If $\mathcal{U} \geq \mathcal{V}$, let $(\mathcal{U} - \mathcal{V}) \in \mathcal{M}_X$ be defined $(\mathcal{U} - \mathcal{V})x \stackrel{\text{def}}{=} \mathcal{U}x - \mathcal{V}x$. Given $k \in \mathbb{N}$ and $\mathcal{U} \in \mathcal{M}_X$, $k\mathcal{U}(x) \stackrel{\text{def}}{=} k \cdot \mathcal{U}(x)$.

$\mathcal{M}X$ is the action on objects of the functor $\mathcal{M}_- : \mathbf{Set} \to \mathbf{Set}$. On functions, $\mathcal{M}f : \mathcal{M}X \to \mathcal{M}Y$ is defined $\mathcal{M}f\mathcal{U}(y) = \sum_{x \in X : f(x)=y} \mathcal{U}x$; note that since $\mathcal{U}$ is nonzero on a finite subset of X, this is well-defined. There is a natural transformation $\mu_X : \mathcal{M}_{\mathcal{M}X} \to \mathcal{M}_X$ that takes $\mu_X\mathcal{V}(x) = \sum_{\mathcal{V}(\mathcal{U}) \geq 0} \mathcal{V}\mathcal{U} \cdot \mathcal{U}x$ and $\eta_X : X \to \mathcal{M}_X$ where $\eta_X x(y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise.} \end{cases}$ It is not difficult to check that $(\mathcal{M}_-, \mu, \eta)$ is a monad, commonly referred to as the multiset monad. Given $f : A \to \mathcal{M}_B$, the definition of $f^\# : \mathcal{M}_A \to \mathcal{M}_B$ follows from a simple calculation: $f^\#(\mathcal{U}) = \sum_{a \in A} (\mathcal{U}a) f(a)$.

Let $\mathbf{Rel}^{\mathcal{M}} \stackrel{\text{def}}{=} Kl(\mathcal{M}_-)$ and $\mathbf{Rel}_f^{\mathcal{M}}$ be the full subcategory of $\mathbf{Rel}^{\mathcal{M}}$ with objects the finite sets. The arrows of $\mathbf{Rel}_f^{\mathcal{M}}$ are thus functions $f : X \to \mathcal{M}_Y$ in $\mathbf{Set}_f$, we will sometimes write $f : X \nrightarrow Y$.

4.1 Multi Synchronisations

Suppose that $f : A \nrightarrow X$ and $g : B \nrightarrow X$ in $\mathbf{Rel}_f^{\mathcal{M}}$. A (multi f, g) *synchronisation* is a pair $(\mathcal{U}, \mathcal{V})$ with $\mathcal{U} \in \mathcal{M}_A$ and $\mathcal{V} \in \mathcal{M}_B$ such that $f^\#\mathcal{U} = g^\#\mathcal{V}$. A synchronisation thus consists of a multiset of A together with a multiset of B that both map to the same multiset of X via $f^\#$ and $g^\#$, respectively; this notion is the multiset equivalent of the notion of synchronisation that we have considered in §2.2. We will again write $\langle \mathcal{U} \curlyvee \mathcal{V} \rangle$ as shorthand and write $\mathsf{snc}(f, g)$ for the set of synchronisations.

Synchronisations inherit an ordering from multisets, pointwise. If we have $\langle \mathcal{U}' \curlyvee \mathcal{V}' \rangle \leq \langle \mathcal{U} \curlyvee \mathcal{V} \rangle$ then $\langle \mathcal{U} - \mathcal{U}' \curlyvee \mathcal{V} - \mathcal{V}' \rangle$: indeed $f^\#(\mathcal{U} - \mathcal{U}') = f^\#\mathcal{U} - f^\#\mathcal{U}' = g^\#\mathcal{V} - g^\#\mathcal{V}' = g^\#(\mathcal{V} - \mathcal{V}')$. Synchronisations are closed under linear combinations: if $\{\langle \mathcal{U}_i \curlyvee \mathcal{V}_i \rangle\}_{i \in I}$ and $k_i \in \mathbb{N}$ then define $\sum_i k_i \langle \mathcal{U}_i \curlyvee \mathcal{V}_i \rangle \stackrel{\text{def}}{=} (\sum_i k_i \mathcal{U}_i, \sum_i k_i \mathcal{V}_i)$, which is clearly a synchronisation.

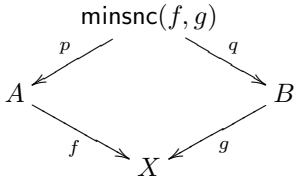
A set $\mathbf{X}$ of synchronisations is *mutually incomparable* when

$$\forall \langle \mathcal{U} \curlyvee \mathcal{V} \rangle, \langle \mathcal{U}' \curlyvee \mathcal{V}' \rangle \in \mathbf{X}.\ \langle \mathcal{U} \curlyvee \mathcal{V} \rangle \leq \langle \mathcal{U}' \curlyvee \mathcal{V}' \rangle \ \vee\ \langle \mathcal{U}' \curlyvee \mathcal{V}' \rangle \leq \langle \mathcal{U} \curlyvee \mathcal{V} \rangle \Rightarrow \langle \mathcal{U} \curlyvee \mathcal{V} \rangle = \langle \mathcal{U}' \curlyvee \mathcal{V}' \rangle.$$

We need to recall a version of Dickson's lemma [8], stated in terms of synchronisations. It can be proved by a straightforward induction.

Lemma 41 (Dickson). *Suppose $f : A \nrightarrow X$ and $g : B \nrightarrow X$ in $\mathbf{Rel}_f^{\mathcal{M}}$. Any set X of mutually incomparable multi-f, g synchronisations is finite.*

Let $\mathsf{minsnc}(f, g)$ be the set of minimal synchronisations. Clearly any two minimal synchronisations are incomparable, thus, by the conclusion of Lemma 41, $\mathsf{minsnc}(f, g)$ is finite. In particular (6) is a commutative diagram in $\mathbf{Rel}_f^{\mathcal{M}}$ where $p\langle \mathcal{U} \curlyvee \mathcal{V} \rangle = \mathcal{U}$ and $q\langle \mathcal{U} \curlyvee \mathcal{V} \rangle = \mathcal{V}$.



(6)

4.2 Weak Pullbacks in $\mathbf{Rel}_f^{\mathcal{M}}$

The following result shows that any synchronisation can be written as a linear combination of minimal synchronisations.

Lemma 42. *If $\langle \mathcal{U} \curlyvee \mathcal{V} \rangle$ then there exists a family $\{(k_i, \langle \mathcal{U}_i \curlyvee \mathcal{V}_i \rangle)\}_{i \in I}$, where each $\langle \mathcal{U}_i \curlyvee \mathcal{V}_i \rangle$ is minimal and different from $\langle \mathcal{U}_j \curlyvee \mathcal{V}_j \rangle$ for all $j \neq i$, s.t. $\langle \mathcal{U} \curlyvee \mathcal{V} \rangle = \sum_i k_i \langle \mathcal{U}_i \curlyvee \mathcal{V}_i \rangle$. The family is called a* minimal decomposition *of $\langle \mathcal{U} \curlyvee \mathcal{V} \rangle$.*

Proof. Simple induction. □

The conclusion of Lemma 42 implies that (6) is a weak pullback diagram: given $\alpha : Y \twoheadrightarrow A$ and $\beta : Y \twoheadrightarrow B$ such that $f\alpha = g\beta$ in $\mathbf{Rel}_f^{\mathcal{M}}$, $h : Y \twoheadrightarrow \mathsf{minsnc}(f, g)$ takes y to a minimal decomposition of $\langle \alpha y \curlyvee \beta y \rangle$.

Remark 43. *The diagram* (6) *is merely a weak pullback, because the decomposition of Lemma 42 is not, in general, unique. Indeed, consider $t : 2 \to 1$ in $\mathbf{Rel}_f^{\mathcal{M}}$ with $t0 = t1 = \{0\}$. Now $\mathsf{minsnc}(t, t) = \{\langle \{0\} \curlyvee \{0\} \rangle, \langle \{0\} \curlyvee \{1\} \rangle, \langle \{1\} \curlyvee \{0\} \rangle, \langle \{1\} \curlyvee \{1\} \rangle\}$. Consider $u : 1 \to 2$ in $\mathbf{Rel}_f^{\mathcal{M}}$ with $u0 = \{0, 1\}$. Then $\langle u0 \curlyvee u0 \rangle$ but there are several minimal decompositions: eg $\langle \{0\} \curlyvee \{0\} \rangle + \langle \{1\} \curlyvee \{1\} \rangle$ and $\langle \{0\} \curlyvee \{1\} \rangle + \langle \{1\} \curlyvee \{0\} \rangle$.*

5 Linking Diagrams in $\mathbf{Spr}(\mathbf{Rel}_f^{\mathcal{M}})$

Consider $\mathsf{Spr}(\mathbf{Rel}_f^{\mathcal{M}})$, with objects that the natural numbers and arrows spans $k \xleftarrow{f} x \xrightarrow{g} l$ in $\mathbf{Rel}_f^{\mathcal{M}}$ where $x \to k \times l$ is injective[6]. Composition proceeds in two steps. First, given

$$k_0 \xleftarrow{f_0} x_0 \xrightarrow{g_0} k_1 \xleftarrow{f_1} x_1 \xrightarrow{g_1} k_2,$$

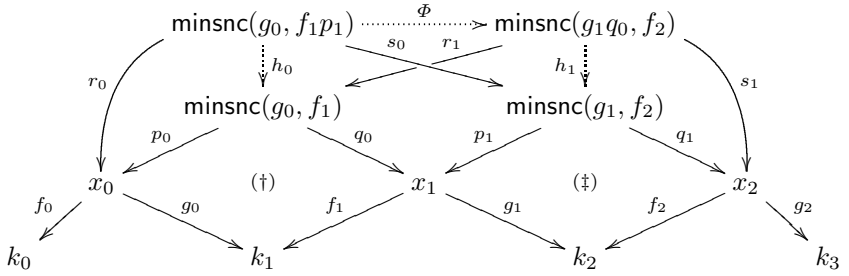
construct $k_0 \xleftarrow{f_0 p} \mathsf{minsnc}(g_0, f_1) \xrightarrow{g_1 q} k_2$ where $p : \mathsf{minsnc}(g_0, f_1) \to x_0$ and $q : \mathsf{minsnc}(g_0, f_1) \to x_1$ are the projections. In general, however, $[f_0 p, g_1 q] : \mathsf{minsnc}(g_0, f_1) \to k_0 \times k_2$ may be non-injective, thus we obtain $\mathsf{minsnc}(g_0, f_1)'$ below, together with f', g' in $\mathbf{Rel}_f^{\mathcal{M}}$ through an epi-mono factorisation of $[f_0{}^{\#} p, g_1{}^{\#} q]$ in **Set**, and this is the composition.

[6] In other words, the internal binary relations in $\mathbf{Rel}_f^{\mathcal{M}}$: an internal relation is a span $k \leftarrow x \to l$ where $x \to k \times l$ is mono.

$$k_0 \xleftarrow{f'} \mathsf{minsnc}(g_0, f_1)' \xrightarrow{g'} k_2$$

Proposition 1. $\mathsf{Spr}(\mathbf{Rel}_f^{\mathcal{M}})$ *is a category.*

Proof. (Sketch) The non-trivial part is showing that composition is associative. The essence is captured in the diagram below, in $\mathbf{Rel}_f^{\mathcal{M}}$.



In addition to the two weak pullback diagrams (†) and (‡), we have a set $\mathsf{minsnc}(g_0, f_1p_1)$ and the projection maps in $\mathbf{Rel}_f^{\mathcal{M}}$

$$r_0 : \mathsf{minsnc}(g_0, f_1p_1) \to x_0,\ \ s_0 : \mathsf{minsnc}(g_0, f_1p_1) \to \mathsf{minsnc}(g_1, f_2)$$

and a set $\mathsf{minsnc}(g_1q_0, f_2)$ together with maps

$$r_1 : \mathsf{minsnc}(g_1q_0, f_2) \to \mathsf{minsnc}(g_0, f_1),\ \ s_1 : \mathsf{minsnc}(g_1q_0, f_2) \to x_2$$

The sets $\mathsf{minsnc}(g_0, f_1p_1)$, $\mathsf{minsnc}(g_1q_0, f_2)$ are not, in general isomorphic, for similar reasons why the $\mathsf{minsnc}(f, g)$ construction fails to be a pullback; there is, in general, more than one decomposition of a synchronisation into a linear combination of minimal synchronisations.

This is not a problem, because all that we require is that $(f_0r_0, g_2q_1s_0)$ and $(f_0p_0r_1, g_2s_1)$ have the same image in $\mathcal{M}_{k_0} \times \mathcal{M}_{k_2}$.

To show this, first we use the weak pullback property of (†) to obtain $h_0 : \mathsf{minsnc}(g_0, f_1p_1) \to \mathsf{minsnc}(g_0, f_1)$, satisfying $p_0h_0 = r_0$ and $q_0h_0 = p_1s_0$. The second of these equations, together with the fact that $\mathsf{minsnc}(g_1q_0, f_2)$ is a weak pullback allows us to obtain

$$\Phi : \mathsf{minsnc}(g_0, f_1p_1) \to \mathsf{minsnc}(g_1q_0, f_2)$$

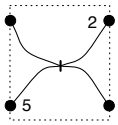
that satisfies $r_1\Phi = h_0$ and $s_1\Phi = q_1s_0$. Now, for any $\sigma \in \mathsf{minsnc}(g_0, f_1p_1)$ we have $f_0r_0\sigma = f_0p_0h_0\sigma = f_0p_0r_1\Phi\sigma$ and $g_2q_1s_0\sigma = g_2s_1\Phi\sigma$, so the image of $(f_0r_0, g_2q_1s_0)$ is contained in the image of $(f_0p_0r_1, g_2s_1)$. A symmetric argument, constructing morphisms $h_1 : \mathsf{minsnc}(g_1q_0, f_2) \to \mathsf{minsnc}(g_1, f_2)$ and $\Psi : \mathsf{minsnc}(g_1q_0, f_2) \to \mathsf{minsnc}(g_0, f_1p_1)$ allows us to demonstrate the reverse inclusion.

□

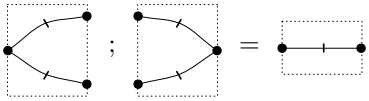
Note that, as indicated in the proof above, the "relational" requirement on spans is necessary in order to ensure associativity of composition. Again there is a tensor product inherited from the coproduct in $\mathbf{Set}_f$.

5.1 The Algebra of $\mathsf{Spr}(\mathbf{Rel}_f^{\mathcal{M}})$

While we no longer have to draw contention, in $\mathsf{Spr}(\mathbf{Rel}_f^{\mathcal{M}})$ links can have multiple connections to boundary ports. We indicate this by annotating connections with natural numbers ≥ 2: for instance the diagram to the right is the span $2 \overset{a}{\nleftarrow} 1 \overset{b}{\nrightarrow} 2$ where $(a0)(0) = (b0)(1) = 1$, $(a0)(1) = 5$ and $(b0)(0) = 2$.

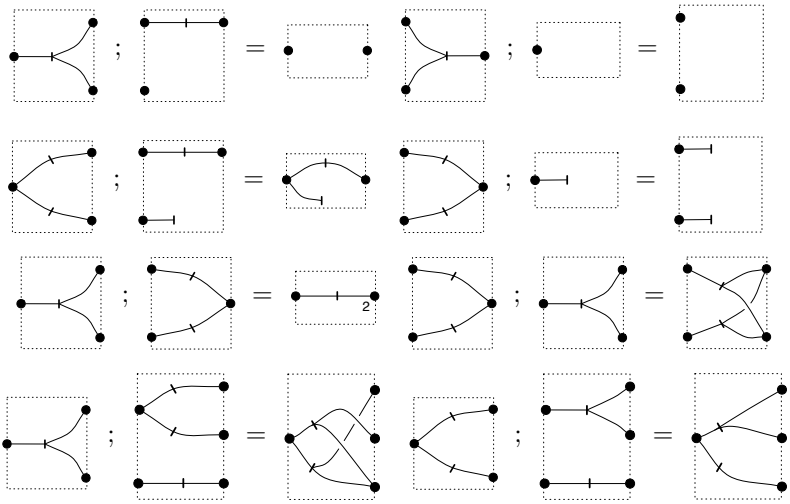


Considering the diagrams of $(\Delta \bot \nabla \top)$ and $(\mathsf{I}\,\mathsf{X})$ in $\mathsf{Spr}(\mathbf{Rel}_f^{\mathcal{M}})$, all the equations in $(\Delta\mathrm{UC})$, $(\Delta\mathrm{A})$, (F), (S), (CC) hold in $\mathsf{Spr}(\mathbf{Rel}_f^{\mathcal{M}})$. On the other hand, the structure in $(\wedge \downarrow \vee \uparrow)$ and $(\mathsf{I}\,\mathsf{X})$ satisfies the equations in $(\wedge\mathrm{UC})$, $(\wedge\mathrm{A})$, (B) and $(\vee\downarrow)$. Differently from $(\wedge\vee)$, in $\mathsf{Spr}(\mathbf{Rel}_f^{\mathcal{M}})$ we have the following:



$(\wedge\vee^{\mathcal{M}})$

Below, we show how $(\Delta \bot \nabla \top)$ and $(\wedge \downarrow \vee \uparrow)$ interact in $\mathsf{Spr}(\mathbf{Rel}_f^{\mathcal{M}})$.



$(\Delta\downarrow\uparrow^{\mathcal{M}})$

$(\wedge\bot\top^{\mathcal{M}})$

$(\Delta\vee^{\mathcal{M}})$

$(\Delta\wedge^{\mathcal{M}})$

The equations in $(\Delta\downarrow\uparrow^{\mathcal{M}})$ are the same as in $(\Delta\downarrow\uparrow^{c})$. The left equation in $(\wedge\bot\top^{\mathcal{M}})$ is the same as the corresponding one in $(\wedge\top\bot^{c})$, but the right hand side equations differ because the contention relation does not play a role in $\mathsf{Spr}(\mathbf{Rel}_f^{\mathcal{M}})$. The right hand side equation in $(\Delta\wedge^{\mathcal{M}})$ agrees with the corresponding one in $(\Delta\vee^{c})$, but the left one deserves attention: while in $(\Delta\vee^{c})$ there was no possible synchronisation between Δ and $\vee$ because of the fact that the two links in $\vee$ were in contention, in $\mathsf{Sp}(\mathbf{Rel}_f^{\mathcal{M}})$ there is a synchronisation that involves all three links, as represented in the left hand side equation of $(\Delta\wedge^{\mathcal{M}})$. The interaction between $\wedge$ and $\wedge$ is as in $(\Delta\wedge^{c})$, and $(\Delta\wedge^{\mathcal{M}})$ are the same as $(\Delta\wedge^{c})$.

Theorem 3. *Every arrow in* $\mathsf{Spr}(\mathbf{Rel}_f^{\mathcal{M}})$ *decomposes into an expression consisting only of* $\Delta, \bot, \nabla, \top, \wedge, \downarrow, \vee, \uparrow, \mathsf{I}, \mathsf{X}$, *composed with* ; *and* $\otimes$.

Proof. Omitted.

6 Conclusion

We have studied two categories of linking diagrams. The first, $\mathsf{Sp}(\mathbf{Rel}_f^c)$, arose from the study of a compositional algebra of C/E nets, called C/E nets with boundaries. Indeed, the arrows of $\mathsf{Sp}(\mathbf{Rel}_f^c)$ are just C/E nets with boundaries, without places. The second, $\mathsf{Spr}(\mathbf{Rel}_f^{\mathcal{M}})$ arose from the study of a compositional algebra of P/T nets, called P/T nets with boundaries. The arrows of $\mathsf{Spr}(\mathbf{Rel}_f^{\mathcal{M}})$ are P/T nets with boundaries, without places. These categories generalise previous work by Hughes [11].

Both categories are "expressive enough" to carry two different commutative monoid-comonoid structures on objects, one of which a separable Frobenius algebra, the other a commutative bialgebra. In both settings the interaction between the two structures is interesting and we have examined some of the phenomena that arise. Both categories are generated by the small number of basic components that witness the monoid-comonoid structures.

In future work a full axiomatisation will be presented, and the categories of linking diagrams will be shown to characterise the arrows of the resulting free categories. The theory of PROPs [16] seems well adapted for expressing the relationship between the algebraic structures, as well as the complete algebras of C/E and P/T nets; Fiore and Campos [9] have recently used a similar setting to develop the algebra of dags.

Acknowledgment. Thanks to R.F.C. Walters for inspiration and guidance, and to the referees for remarks that have improved the presentation.

References

1. Abramsky, S., Coecke, B.: A categorical semantics of quantum protocols. In: Logic in Computer Science (LiCS 2004). IEEE Press (2004)
2. Arbab, F.: Reo: A channel-based coordination model for component composition. Math. Struct. Comp. Sci. 14(3), 1–38 (2004)
3. Bruni, R., Lanese, I., Montanari, U.: A basic algebra of stateless connectors. Theor. Comput. Sci. 366, 98–120 (2006)
4. Bruni, R., Melgratti, H., Montanari, U.: A connector algebra for P/T nets interactions. In: Katoen, J.-P., König, B. (eds.) CONCUR 2011. LNCS, vol. 6901, pp. 312–326. Springer, Heidelberg (2011)
5. Bruni, R., Melgratti, H.C., Montanari, U., Sobociński, P.: Connector algebras for C/E and P/T nets' interactions. Log. Meth. Comput. Sci. (to appear, 2013)
6. Carboni, A., Walters, R.F.C.: Cartesian bicategories I. J. Pure Appl. Algebra 49, 11–32 (1987)

7. Coecke, B., Paquette, É.O., Pavlovic, D.: Classical and quantum structuralism. In: Semantical Techniques in Quantum Computation, pp. 29–69. Cambridge University Press (2009)
8. Dickson, L.E.: Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. Amer. Journal Math. 35(4), 413–422 (1913)
9. Fiore, M.P., Campos, M.D.: The algebra of directed acyclic graphs. In: Coecke, B., Ong, L., Panangaden, P. (eds.) Computation, Logic, Games and Quantum Foundations. LNCS, vol. 7860, pp. 37–51. Springer, Heidelberg (2013)
10. Girard, J.-Y.: Linear logic. Theor. Comput. Sci. 50, 1–102 (1987)
11. Hughes, D.J.D.: Linking diagrams for free. arXiv:0805.1441v1 (2008)
12. Joyal, A., Street, R.: The geometry of tensor calculus, i. Adv. Math. 88, 55–112 (1991)
13. Kelly, G.M., Laplaza, M.L.: Coherence for compact closed categories. J. Pure Appl. Algebra 19, 193–213 (1980)
14. Kissinger, A.: Synthesising graphical theories. arxiv.org:1202.6079 (2012)
15. Kock, J.: Frobenius algebras and 2D topological quantum field theories. Cambridge University Press (2003)
16. Lack, S.: Composing PROPs. Theor. App. Categories 13(9), 147–163 (2004)
17. Mac Lane, S.: Categorical algebra. Bull. Amer. Math. Soc. 71, 40–106 (1965)
18. Selinger, P.: Dagger compact closed categories and completely positive maps. In: Quantum Programming Languages (QPL 2007). ENTCS, vol. 170, pp. 139–163 (2007)
19. Selinger, P.: A survey of graphical languages for monoidal categories. arXiv:0908.3347v1 (math.CT) (2009)
20. Sobociński, P.: Representations of petri net interactions. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 554–568. Springer, Heidelberg (2010)
21. Sobociński, P., Stephens, O.: Reachability via compositionality in Petri nets. arXiv:1303.1399v1 (2013)
22. Sobociński, P., Stephens, O.: Penrose: Putting compositionality to work for petri net reachability. In: Heckel, R., Milius, S. (eds.) CALCO 2013. LNCS, vol. 8089, pp. 346–352. Springer, Heidelberg (2013)

A Logic-Programming Semantics of Services

Ionuţ Ţuţu[1,2] and José Luiz Fiadeiro[1]

[1] Dept. Computer Science, Royal Holloway University of London, UK
[2] Institute of Mathematics of the Romanian Academy,
Research group of the project ID-3-0439
ittutu@gmail.com, jose.fiadeiro@rhul.ac.uk

Abstract. We develop formal foundations for notions and mechanisms needed to support service-oriented computing. Our work provides semantics for the service overlay by abstracting concepts from logic programming. It draws a strong analogy between the discovery of a service that can be bound to a client application and the search for a clause that can be used for computing an answer to a query. In addition, it describes the process of binding services and the reconfiguration of applications as service-oriented derivatives of unification and resolution.

1 Introduction

Service-Oriented Computing. SOC is a recent paradigm that addresses computation in 'global computers' – computational infrastructures available globally in which software applications can discover and bind dynamically, at run time, to services offered by providers. Whereas the paradigm has been effectively in use for a more than a decade in the form of Web services [1] or Grid computing [13], research into its formal foundations has lagged somewhat behind, partly because of our lack of understanding of (or agreement on) what is really new about the paradigm.

It is fair to say that significant advances have been made towards formalising new forms of distributed computation that have arisen around the notion of service (e.g. choreography [18]), notably through several variants of the π-calculus. However, SOC raises more profound challenges at the level of the structure of systems thanks to their ability to discover and bind dynamically, in a non-programmed way, to other systems. The structure of the systems we are now creating in the virtual space of computational networks is intrinsically dynamic, a phenomenon hitherto unknown. Formalisms such as the π-calculus do not address these structural properties of systems.

Towards that end, we have investigated algebraic structures that account for modularity (e.g. [10,12]) – the way services are orchestrated as composite structures of components and how binding is performed through interaction protocols – and the mechanisms through which discovery can be formalised in terms of logical specifications of required/provided services and constraint optimisation for service-level agreements (e.g. [9,11]). In the present paper, we take further this research to address the operational aspects behind *dynamic* discovery and binding, i.e. the mechanisms through which applications discover and bind, at run time, to services. Our aim is to develop an abstract, foundational setting – i.e. independent of the specific technologies that are currently deployed, such as SOAP for message-exchange protocols and the UDDI for

R. Heckel and S. Milius (Eds.): CALCO 2013, LNCS 8089, pp. 299–313, 2013.

description, discovery, and integration – that combines both *declarative* and *dynamic* semantics of services. The challenge here is to develop an integrated algebraic framework that accounts for (*a*) logical specifications of services, (*b*) the way models of those specifications capture orchestrations of components that depend on externally provided services to be discovered, and (*c*) the way the discovery of services and the binding of their orchestrations to client applications can be expressed in logical/algebraic terms.

Logic Programming. The approach that we propose to develop to meet this challenge builds on (Horn-clause) logic programming (LP) – the paradigm that epitomises the integration of declarative and operational aspects of logic. In LP, clauses have a declarative semantics as disjunctions of literals that express relationships over a domain (the Herbrand universe), and an operational semantics that derives from resolution and term unification: definite clauses (from a program) are used to resolve queries (expressed as goal clauses) by generating new queries and, through term unification, computing partial answers as substitutions for the variables of the original query.

In a nutshell, the analogy with SOC that we propose to develop works as follows:

- The Herbrand universe consists of all possible service orchestrations with no dependencies on external services ('ground services').
- Variables correspond to requires-points of orchestrations, i.e. dependencies on external services that need to be discovered.
- Terms correspond to services delivered through provides-points of orchestrations.
- Definite clauses express properties of provides-points (head) and requires-points (body) of service orchestrations – what in [11] we call service modules. Their declarative semantics is that, when bound to applications that deliver services satisfying the properties of the requires-points, the orchestrations will deliver at the provides-points services that satisfy the specified properties.
- Goal clauses express properties of orchestrations of services that an application requires in order to fulfil given business goals – what in [11] we call activity modules.
- Programs correspond to service repositories.
- Resolution and term unification account for service discovery by matching required properties with provided ones and the binding of required with provided services.

Structure of the Paper. In Sect. 2 we propose an algebraic model of service orchestrations as asynchronous relational networks similar to those used in [8], and we define the logical framework over which we can express properties of the interaction points through which such networks can be interconnected. We prove that the resulting logic constitutes an institution [14], which provides the declarative semantics of our approach to SOC. In Sect. 3 we show how clauses, unification and resolution can be defined over that institution, thus providing the corresponding operational semantics of SOC.

2 Asynchronous Relational Networks

Our first contribution is the formulation of an institution of asynchronous relational networks. This accounts for the service-oriented counterpart of the declarative aspects of conventional LP within first-order logic, in particular the key role played by variables – the structures over which the computational aspects of LP operate.

The concepts discussed here depend upon elements of linear temporal logic (LTL). However, the proposed theory is largely independent of the logical framework of choice, and can be easily adapted to any institution such that

- the category of signatures is (finitely) co-complete;
- there exist co-free models along any signature morphism, i.e. the reduct functor of any signature morphism has a right adjoint;
- the category of models of any signature has products;
- any model homomorphism reflects the satisfaction of any sentence.

In order to capture a more operational notion of service orchestration, we work with a variant MA-LTL of LTL whose models are not traces, but Muller automata [17]; by definition, an automaton satisfies a sentence if and only if every trace accepted by the automaton satisfies the considered sentence.

2.1 Signatures and Signature Morphisms

We start by defining the category $\mathbb{A}RN$ of signatures and signature morphisms of our institution. These are asynchronous relational networks similar to those defined in [8] except that we use Muller automata instead of sets of traces as models of behaviour, and hypergraphs instead of graphs.

Following [8], we regard service components as networks of processes that interact asynchronously by exchanging messages through communication channels. Messages are considered to be atomic units of communication. They can be grouped into structures – ports – through which processes and channels can be interconnected.

Ports are sets of messages with attached polarities. As in [2,3] we distinguish between outgoing or published messages (labelled with a minus sign) and incoming or delivered messages (labelled with a plus sign).

Definition 1 (Port). *A* port M *is a pair* $\langle M^-, M^+ \rangle$ *of disjoint finite sets of messages. The set of all* messages *of* M *is given by the union* $M^- \cup M^+$*, usually denoted by* M*. Every port* M *determines the set of* actions $A_M = A_{M^-} \cup A_{M^+}$*, where* A_{M^-} *is the set* $\{m! \mid m \in M^-\}$ *of* publications*, and* A_{M^+} *is the set* $\{m¡ \mid m \in M^+\}$ *of* deliveries*.*

Processes are defined by sets of interaction points labelled with ports and by Muller automata that describe process behaviour in terms of observable actions.

Definition 2 (Process). *A* process $(\{M_x \mid x \in X\}, \Lambda)$ *consists of a finite set* X *of* interaction points*, each point* $x \in X$ *being labelled with a port* M_x*, and a Muller automaton* Λ *over the alphabet* $\mathcal{P}(A_M)$*, where* M *is the port given by*

$$M^{\mp} = \biguplus_{x \in X} M_x^{\mp} = \{x.m \mid x \in X, m \in M_x^{\mp}\} \ .$$

Example 1. In Fig. 1 we depict a process JourneyPlanner that provides directions from a source to a target location. The process interacts with the environment through two ports: JP_1 and JP_2. The first port is used for communicating with potential client processes – the request for directions (including the source and the target locations) is encoded into the incoming message planJourney, while the response is represented by the

outgoing message directions. The second port defines messages that JourneyPlanner exchanges with other processes in order to complete its task – the outgoing message getRoutes can be seen as a query for all possible routes between the specified source and target locations, and the incoming messages routes and timetables define the result of the query and the timetables of the available transport services for the selected routes.

Fig. 1. The JourneyPlanner process

The behaviour of JourneyPlanner is given by the Muller automaton depicted in Fig. 2, whose final state sets contain q_0 whenever they contain q_5. We can describe it informally as follows: whenever JourneyPlanner receives a request planJourney it immediately initiates the search of the available routes by sending the message getRoutes; it then waits for the delivery of the routes and of the corresponding timetables; once it receives both it compiles the directions and replies to the client.

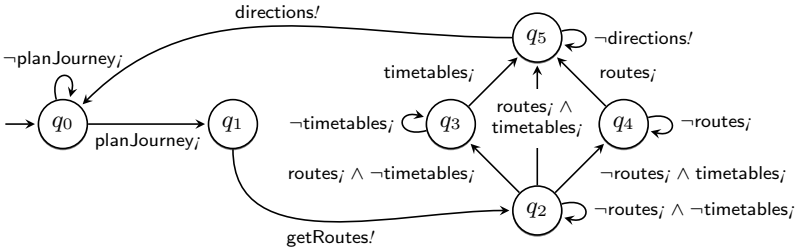


Fig. 2. The JourneyPlanner automaton[1]

Note that every polarity-preserving map θ between ports M and M' defines a function $A_\theta \colon A_M \to A_{M'}$, often denoted simply by θ, that maps every publication action $m!$ into $\theta(m)!$ and every delivery action $m¡$ into $\theta(m)¡$.

Fact 1. For any process $(\{M_x \mid x \in X\}, \Lambda)$, the injections $\{x\_ \colon A_{M_x} \to A_M\}_{x \in X}$ define a co-product in the category of MA-LTL-signatures.

Processes communicate by transmitting messages through channels. As in [3,8], channels are bidirectional: they may transmit both incoming and outgoing messages.

Definition 3 (Channel). *A* channel (M, Λ) *consists of a finite set* M *of* messages *and a Muller automaton* Λ *over the alphabet* $\mathcal{P}(A_M)$*, where* A_M *is the union* $A_M^- \cup A_M^+$ *of* $A_M^- = \{m! \mid m \in M\}$ *and* $A_M^+ = \{m¡ \mid m \in M\}$.

[1] In the graphical representation the transitions are labelled with propositional sentences; this means that there exists a transition for any model (set of actions) of the considered sentence.

In order to enable given processes to exchange messages, channels need to be attached to their ports, thus forming connections.

Definition 4 (Connection). *A* connection $(\{\mu_x \colon M \rightharpoonup M_x \mid x \in X\}, \Lambda)$ *between the ports* $\{M_x \mid x \in X\}$ *consists of a channel* (M, Λ) *and a finite family of partial* attachment injections $\{\mu_x \colon M \rightharpoonup M_x \mid x \in X\}$ *such that* $M = \bigcup_{x \in X} \mathrm{dom}(\mu_x)$ *and* $\mu_x^{-1}(M_x^{\mp}) \subseteq \bigcup_{y \in X \setminus \{x\}} \mu_y^{-1}(M_y^{\pm})$*, for any* $x \in X$.

This notion of connection differs from the one found in [8] in that messages can be transmitted between more than two ports. The additional condition ensures that messages are well paired: every published message of M_x, for $x \in X$, is paired with a delivered message of M_y, for $y \in X \setminus \{x\}$, and vice versa.

Example 2. In order to illustrate how the process JourneyPlanner can send or receive messages, we consider the connection C depicted in Fig. 3 that moderates the flow of messages between the port JP_2 and two other ports, R_1 and R_2.

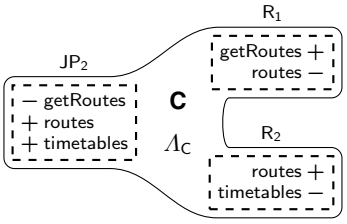


Fig. 3. The JourneyPlanner's connection

The underlying channel of C is given by the set of messages $M = \{g, r, t\}$ together with the automaton Λ_{C} that specifies the delivery of all published messages without any delay. Λ_{C} can be built as the product of the automata Λ_m, for $m \in M$, whose transition map is depicted in Fig. 4, and whose sets of states are all marked as final.

The channel is attached to the ports JP_2, R_1 and R_2 through the partial injections:

- $\mu_{\mathsf{JP}_2} = \{g \mapsto \mathsf{getRoutes}, r \mapsto \mathsf{routes}, t \mapsto \mathsf{timetables}\}$,
- $\mu_{\mathsf{R}_1} = \{g \mapsto \mathsf{getRoutes}, r \mapsto \mathsf{routes}\}$ and
- $\mu_{\mathsf{R}_2} = \{r \mapsto \mathsf{routes}, t \mapsto \mathsf{timetables}\}$.

Note that the actual senders and receivers of messages are specified through the attachment injections. For example, g is delivered only to the port R_1 (because μ_{R_2} is not defined on g), while r is simultaneously delivered to both JP_2 and R_2.

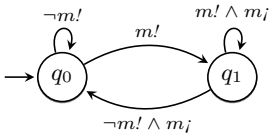


Fig. 4. The Λ_m automaton

As already suggested in Ex. 1 and 2, processes and connections have dual roles and interpret the polarities of the messages accordingly. In this sense, processes are responsible for publishing messages (i.e. delivered messages are inputs and published messages are outputs), while connections are responsible for delivering messages.

We clarify this dual nature of connections $(\{\mu_x \colon M \rightharpoonup M_x \mid x \in X\}, \Lambda)$ by defining partial translations $\{A_{\mu_x} \colon A_M \rightharpoonup A_{M_x} \mid x \in X\}$ given by

- $\mathrm{dom}(A_{\mu_x}) = \{m! \mid m \in \mu_x^{-1}(M_x^-)\} \cup \{m¡ \mid m \in \mu_x^{-1}(M_x^+)\}$,
- $A_{\mu_x}(m!) = \mu_x(m)!$ for all $m \in \mu_x^{-1}(M_x^-)$,
- $A_{\mu_x}(m¡) = \mu_x(m)¡$ for all $m \in \mu_x^{-1}(M_x^+)$.

We often designate the partial maps A_{μ_x} simply by μ_x if there is no risk of confusion.

Fact 2. Every connection $(\{\mu_x \colon M \rightharpoonup M_x \mid x \in X\}, \Lambda)$ defines a family of spans $\left\{A_M \xleftarrow{\supseteq} \mathrm{dom}(\mu_x) \xrightarrow{\mu_x} A_{M_x}\right\}_{x \in X}$ in the category of MA-LTL-signatures.

We can now define asynchronous networks of processes as hypergraphs having vertices labelled with ports and hyperedges labelled with processes and connections.

Definition 5 (Hypergraph). *An* (edge-labelled) hypergraph (X, E, γ) *consists of a set* X *of* vertices *or* nodes*, a set* E *of* hyperedges *disjoint from* X*, and an* incidence map $\gamma \colon E \to \mathcal{P}(X)$ *defining for every hyperedge* $e \in E$ *a non-empty set* $\gamma_e \subseteq X$ *of vertices it is incident with. A hypergraph* (X, E, γ) *is said to be* edge-bipartite *if* E *is partitioned into two subsets* F *and* G *such that no adjacent hyperedges belong to the same partition, i.e. for every two hyperedges* $e_1, e_2 \in E$ *such that* $\gamma_{e_1} \cap \gamma_{e_2} \neq \emptyset$*, either* $e_1 \in F$ *and* $e_2 \in G$*, or* $e_1 \in G$ *and* $e_2 \in F$*.*

Hypergraphs have been used extensively in the context of graph-rewriting-based approaches to concurrency, including SOC (e.g., [4,7]). We use them instead of graphs [8] because they offer a more flexible mathematical framework for handling the notions of variable and variable binding that we require in Sect. 3.

Definition 6 (Asynchronous Relational Network – ARN). *An* asynchronous relational network $\alpha = (X, P, C, \gamma, M, \mu, \Lambda)$ *consists of a (finite) edge-bipartite hypergraph* (X, P, C, γ) *of* points $x \in X$*,* computation hyperedges $p \in P$ *and* communication hyperedges $c \in C$*, together with*

- *a port* M_x *for every point* $x \in X$*,*
- *a process* $(\{M_x \mid x \in \gamma_p\}, \Lambda_p)$ *for every hyperedge* $p \in P$*, and*
- *a connection* $(\{\mu_x^c \colon M_c \rightharpoonup M_x \mid x \in \gamma_c\}, \Lambda_c)$ *for every hyperedge* $c \in C$*.*

Example 3. By putting together the process and the connection presented in Ex. 1 and 2, we obtain the ARN JourneyPlanner depicted in Fig. 5. Its underlying hypergraph consists of the points JP_1, JP_2, R_1 and R_2, the computation hyperedge JP, the communication hyperedge C, and the incidence map γ given by $\gamma_{\mathsf{JP}} = \{\mathsf{JP}_1, \mathsf{JP}_2\}$ and $\gamma_{\mathsf{C}} = \{\mathsf{JP}_2, \mathsf{R}_1, \mathsf{R}_2\}$.

An *interaction-point* of an ARN α is a point of α that is not bound to both computation and communication hyperedges. We distinguish between requires-points and provides-points, as follows.

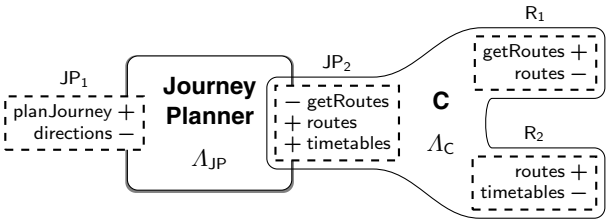


Fig. 5. The JourneyPlanner ARN

Definition 7 (Requires and Provides-point). *A* requires-point *of an ARN α is a point of α that is incident only with a communication hyperedge. Similarly, a* provides-point *of α is a point incident only with a computation hyperedge.*

Morphisms of ARNs can be defined as injective homomorphisms between their underlying hypergraphs that preserve all labels, except those associated with requires-points.

Definition 8 (Homomorphism of Hypergraphs). *A* homomorphism h *between hypergraphs (X, E, γ) and (X', E', γ') consists of functions $h^v \colon X \to X'$ and $h^e \colon E \to E'$ such that for any $x \in X$ and $e \in E$, $x \in \gamma_e$ if and only if $h^v(x) \in \gamma'_{h^e(e)}$.*

Definition 9 (Morphism of ARNs). *Given two ARNs $\alpha = (X, P, C, \gamma, M, \mu, \Lambda)$ and $\alpha' = (X', P', C', \gamma', M', \mu', \Lambda')$, a* morphism $\delta \colon \alpha \to \alpha'$ *consists of*

- *an injective homomorphism $\delta \colon (X, P, C, \gamma) \to (X', P', C', \gamma')$ between the underlying hypergraphs of α and α' such that $\delta^e(P) \subseteq P'$ and $\delta^e(C) \subseteq C'$, and*
- *a family of polarity-preserving injections $\delta^{pt} = \left\{\delta^{pt}_x \colon M_x \to M'_{\delta^v(x)}\right\}_{x \in X}$,*

such that

- *for every non-requires-point $x \in X$, $\delta^{pt}_x = 1_{M_x}$,*
- *for every computation hyperedge $p \in P$, $\Lambda_p = \Lambda'_{\delta^e(p)}$, and*
- *for every communication hyperedge $c \in C$, $M_c = M'_{\delta^e(c)}$, $\Lambda_c = \Lambda'_{\delta^e(c)}$ and the following diagram commutes, for any point $x \in \gamma_c$.*

$$\begin{array}{ccc} M_c = M'_{\delta^e(c)} & \xrightarrow{\mu^c_x} & M_x \\ & \searrow^{(\mu')^{\delta^e(c)}_{\delta^v(x)}} & \downarrow{\scriptstyle \delta^{pt}_x} \\ & & M'_{\delta^v(x)} \end{array}$$

Proposition 1. *The morphisms of ARNs form a category, denoted $\mathbb{A}RN$, in which the composition is defined component-wise, with left and right identities given by morphisms whose components are set-theoretic identities.*

2.2 Sentences and Sentence Translations

We now define the sentence functor of our institution.

Definition 10 (Sentence). *For any ARN α, i.e. for any signature α, the set* $\mathsf{Sen}^{\underline{\mathrm{SOC}}}(\alpha)$ *of* (atomic) α-sentences *is defined as the set of pairs (x, ρ), usually denoted $@_x\,\rho$, where x is a point of α and ρ is an* MA-$\underline{\mathrm{LTL}}$*-sentence over A_{M_x}.*

The *translation* of sentences is straightforward: for every morphism $\delta\colon \alpha \to \alpha'$ of ARNs, the map $\mathsf{Sen}^{\underline{\mathrm{SOC}}}(\delta)\colon \mathsf{Sen}^{\underline{\mathrm{SOC}}}(\alpha) \to \mathsf{Sen}^{\underline{\mathrm{SOC}}}(\alpha')$ is given by

$$\mathsf{Sen}^{\underline{\mathrm{SOC}}}(\delta)(@_x\,\rho) = @_{\delta^v(x)}\,\delta_x^{pt}(\rho)$$

for any point x of α and any MA-$\underline{\mathrm{LTL}}$-sentence ρ over the actions of x.

Proposition 2. $\mathsf{Sen}^{\underline{\mathrm{SOC}}}$ *is a functor $\mathbb{A}RN \to \mathbb{S}et$.*

2.3 Models and Model Reductions

The model functor of our institution assigns ground ARNs to the requires-points of the considered networks.

Definition 11 (Ground ARN). *An ARN is said to be* ground *if it has no requires-points. We denote by $\mathbb{GA}RN$ the full subcategory of $\mathbb{A}RN$ determined by ground ARNs.*

Definition 12 (Model). *For any ARN α, the category* $\mathsf{Mod}^{\underline{\mathrm{SOC}}}(\alpha)$ *of* α-models *or* α-interpretations *is the comma category $\alpha/\mathbb{GA}RN$.*

It follows that α-interpretations are morphisms of ARNs $\iota\colon \alpha \to \beta$ such that β is a ground network, which can also be seen as collections of ground ARNs that are designated to the requires-points of α. In order to explain this in more detail let us introduce the following notions of dependency and ARN defined by a point.

Definition 13 (Dependency). *Let x and y be points of an ARN α. x is said to be* dependent *on y if there exists a path from x to y that begins with a computation hyperedge, i.e. if there exists an alternating sequence $x\, e_1\, x_1 \cdots e_n\, y$ of (distinct) points and hyperedges such that $x \in \gamma_{e_1}$, $y \in \gamma_{e_n}$, $x_i \in \gamma_{e_i} \cap \gamma_{e_{i+1}}$ for any $1 \leq i < n$, and $e_1 \in P$.*

Definition 14 (ARN Defined by a Point). *The* ARN defined by a point x *of an ARN α is the full sub-ARN α_x of α determined by x and the points on which x is dependent.*

One can now see that any interpretation $\iota\colon \alpha \to \beta$ of an ARN α assigns to each requires-point x of α the ground sub-ARN $\beta_{\iota^v(x)}$ of β defined by $\iota^v(x)$.

Example 4. Based on the ground ARN depicted in Fig. 6 we can define an interpretation $\iota\colon \mathrm{JourneyPlanner} \to \mathrm{JourneyPlannerNet}$ that preserves all the labels, points and hyperedges of $\mathrm{JourneyPlanner}$, with the exception of the points R_1 and R_2, which are mapped to MS_1 and TS_1, respectively. In this case, the point MS_1 only depends on itself, hence the sub-ARN of $\mathrm{JourneyPlannerNet}$ defined by MS_1, i.e. the ground ARN assigned to the requires-point R_1 of $\mathrm{JourneyPlanner}$, is given by the process MS and its port MS_1. In contrast, the point JP_1 depends on all other points of $\mathrm{JourneyPlannerNet}$, and thus it defines the entire ARN $\mathrm{JourneyPlannerNet}$.

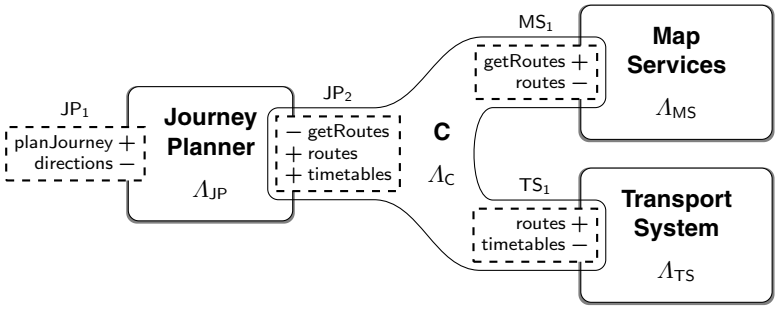


Fig. 6. The JourneyPlannerNet ARN

The reduction of interpretations is defined as the left composition with the considered ARN morphism. For every morphism of ARNs $\delta\colon \alpha \to \alpha'$, the reduct functor $\mathsf{Mod}^{\underline{\mathrm{SOC}}}(\delta)$ is just the composition functor $\delta/\mathbb{GA}RN\colon \alpha'/\mathbb{GA}RN \to \alpha/\mathbb{GA}RN$ given by $(\delta/\mathbb{GA}RN)(\iota') = \delta;\iota'$ and $(\delta/\mathbb{GA}RN)(\zeta') = \zeta'$ for every α'-interpretation ι' and every α'-interpretation homomorphism ζ'.

Proposition 3. $\mathsf{Mod}^{\underline{\mathrm{SOC}}}$ *is a contravariant functor* $\mathbb{A}RN^{\,op} \to \mathbb{C}at$.

2.4 The Satisfaction Relation

The evaluation of ARN sentences with respect to ARN interpretations relies on the concepts of diagram of a network and of automaton defined by a point, whose purpose is to describe the observable behaviour of a ground ARN through one of its points. We start by extending Facts 1 and 2 to ARNs.

Fact 3 (Diagram of an ARN). Every ARN $\alpha = (X, P, C, \gamma, M, \mu, \Lambda)$ defines a diagram $D_\alpha\colon \mathbb{J}_\alpha \to \mathbb{S}ig^{\text{MA-}\underline{\mathrm{LTL}}}$ as follows:

- $\mathbb{J}_\alpha$ is the free preordered category given by the set of objects

$$X \cup P \cup C \cup \{\langle c, x, \alpha\rangle \mid c \in C, x \in \gamma_c\}$$

  and the arrows
  - $\{x \to p \mid p \in P, x \in \gamma_p\}$ for computation hyperedges, and
  - $\{c \leftarrow \langle c, x, \alpha\rangle \to x \mid c \in C, x \in \gamma_c\}$ for communication hyperedges;
- D_α is the functor that provides the sets of actions of ports, processes and channels, together with the appropriate mappings between them. For example, given a communication hyperedge $c \in C$ and a point $x \in \gamma_c$,
  - $D_\alpha(c) = A_{M_c}$, $D_\alpha(\langle c, x, \alpha\rangle) = \mathrm{dom}(\mu^c_x)$, $D_\alpha(x) = A_{M_x}$,
  - $D_\alpha(\langle c, x, \alpha\rangle \to c) = \big(\mathrm{dom}(\mu^c_x) \subseteq A_{M_c}\big)$, and
  - $D_\alpha(\langle c, x, \alpha\rangle \to x) = \mu^c_x$.

Because the category $\mathbb{S}ig^{\text{MA-}\underline{\mathrm{LTL}}}$ is finitely co-complete, we can define the signature of an ARN based on its diagram.

Definition 15 (Signature of an ARN). *The signature of an ARN α is the co-limiting co-cone $\xi\colon D_\alpha \Rightarrow A_\alpha$ of the diagram D_α.*

The most important construction that allows us to define the satisfaction relation is the one that defines the observed behaviour of a (ground) network at one of its points.

Definition 16 (Automaton Defined by a Point). *Let x be a point of a ground ARN β. The* observed automaton *Λ_x at x is given by the reduct $\Lambda_{\beta_x}\restriction_{\xi_x}$, where*

- *$\beta_x = (X, P, C, \gamma, M, \mu, \Lambda)$ is the sub-ARN of β defined by x,*
- *$\xi\colon D_{\beta_x} \Rightarrow A_{\beta_x}$ is the signature of β_x,*
- *Λ_{β_x} is the product automaton $\prod_{e\,\in\, P\cup C} \Lambda_e^{\beta_x}$, and*
- *$\Lambda_e^{\beta_x}$ is the co-free expansion of Λ_e along ξ_e, for any hyperedge $e \in P \cup C$.*

Example 5. Let us consider the ground ARN outlined in Fig. 6. The automaton defined by the point MS_1 is just $\Lambda_{\mathsf{MS}}\restriction_{A_{\mathsf{MS}_1}}$ – this follows from the observation that the ARN defined by MS_1 consists solely of the process MS and the port MS_1. On the other hand, the calculation of the automaton defined by provides-point JP_1 involves the product of the co-free expansions of all four automata Λ_{JP}, Λ_{MS}, Λ_{TS} and Λ_{C}.

We now have all the necessary concepts for defining the satisfaction of ARN sentences by ARN interpretations. Let us thus consider an ARN α, an α-interpretation $\iota\colon \alpha \to \beta$ and an α-sentence $@_x\,\rho$. Then

$$\iota \models^{\mathrm{SOC}}_{\alpha} @_x\,\rho \quad \text{if and only if} \quad \Lambda_{\iota^v(x)}\restriction_{\iota^{pt}_x} \models^{\text{MA-}\underline{\mathrm{LTL}}} \rho \ ,$$

where $\Lambda_{\iota^v(x)}$ is the observed automaton at $\iota^v(x)$ in β.

The construction of the institution of ARNs is completed by the following result, which states that satisfaction is invariant with respect to changes of ARNs.

Proposition 4. *For every ARN morphism $\delta\colon \alpha \to \alpha'$, any α'-interpretation ι' and any α-sentence $@_x\,\rho$,*

$$\iota' \models^{\mathrm{SOC}}_{\alpha'} \mathsf{Sen}^{\mathrm{SOC}}(\delta)(@_x\,\rho) \quad \text{if and only if} \quad \mathsf{Mod}^{\mathrm{SOC}}(\delta)(\iota') \models^{\mathrm{SOC}}_{\alpha} @_x\,\rho \ .$$

Corollary 1. $\underline{\mathrm{SOC}} = \big(\mathbb{A}RN, \mathsf{Sen}^{\underline{\mathrm{SOC}}}, \mathsf{Mod}^{\underline{\mathrm{SOC}}}, \models^{\underline{\mathrm{SOC}}}\big)$ *is an institution.*

3 A Logical View on Service Discovery and Binding

Building on the results of Sect. 2, we now investigate how the semantics of the service overlay can be characterised using fundamental computational aspects of the LP paradigm such as unification and resolution.

Our approach is built upon a simple and intuitive analogy between the SOC concepts of service module and client application [11] and the LP concepts of clause and query [16]. In order to clarify this analogy we rely on the institution $\underline{\mathrm{FOL}}$ of first-order logic [6] and also on the studies on internal logic developed in [19] and [5].

We begin by briefly describing the structure that provides the basic elements involved in defining the denotational and operational semantics of relational LP – the institution of (sets of) variables and substitutions over a first-order signature (S, F, P).

The signatures of this institution are finite sets (or blocks) of variables, i.e. finite sets of pairs (x, s), where x is the name of the variable (distinct from the names of other

variables) and $s \in S$ is its sort. The models, sentences, and the satisfaction relation are inherited from FOL. In this sense, for every set of variables X, we consider the corresponding category of models, set of sentences and satisfaction relation of the extended first-order signature $(S, F \cup X, P)$.

The morphisms of signatures $X \to Y$ are substitutions, i.e. mappings of the variables of X into terms over Y. Based on the evaluation of terms in models and on the canonical extension of substitutions from variables to terms, the substitutions define appropriate reductions of models and translations of sentences, about which it has been shown in [5] that the satisfaction condition holds.

3.1 The Clausal Structure of Services

Given the above constructions, we can describe definite first-order clauses as structures

$$C \xleftarrow[X]{} H$$

such that X is a block of variables, C is a relational atom over X, i.e. a relational atom of the extended signature $(S, F \cup X, P)$, and H is a finite set of relational atoms over X. Their semantics is given by the class of (S, F, P)-algebras whose expansions to $(S, F \cup X, P)$ satisfy C whenever they satisfy every sentence in H. Note that in traditional LP the symbols of variables are often distinguished from other symbols through notational conventions. For this reason, the block X of variables is at times omitted.

Service clauses can be defined in a similar manner, essentially by replacing the institution of first-order substitutions with the institution of ARNs. Intuitively, this means that we replace blocks of variables with ARNs, variables with requires-points, and terms (over variables) with provides-points.

Definition 17 (Clause). *A* SOC-clause *is a structure* (P, α, R)*, also written*

$$P \xleftarrow[\alpha]{} R$$

such that α *is an ARN,* P *is an* α*-sentence referring to a provides-point of* α *and* R *is a finite set of* α*-sentences referring to distinct requires-points of* α*.*

The semantics of service clauses is defined just as the semantics of first-order clauses, except that they are evaluated over the class $\mathcal{G}$ of ground ARNs instead of (S, F, P) ground terms. In this sense, $\mathcal{G}$ (which may be intuitively regarded as the Herbrand universe) satisfies a clause (P, α, R) if and only if any interpretation of α that satisfies all sentences in R satisfies P as well.

Example 6. The ARN JourneyPlanner introduced in Ex. 3 can orchestrate a service module that consistently delivers the requested directions, provided that the routes and the timetables can always be obtained. We specify this through the service clause

$$@_{\mathsf{JP}_1}\, \rho^{\mathrm{JP}} \xleftarrow[\mathrm{JourneyPlanner}]{} \left\{ @_{\mathsf{R}_1}\, \rho_1^{\mathrm{JP}}, @_{\mathsf{R}_2}\, \rho_2^{\mathrm{JP}} \right\}$$

where ρ^{JP}, ρ_1^{JP}, ρ_2^{JP} are the MA-LTL-sentences $\Box\,(\mathsf{planJourney}_{\mathsf{i}} \supset \Diamond\, \mathsf{directions}!)$, $\Box\,(\mathsf{getRoutes}_{\mathsf{i}} \supset \Diamond\, \mathsf{routes}!)$ and $\Box\,(\mathsf{routes}_{\mathsf{i}} \supset \Diamond\, \mathsf{timetables}!)$, respectively.

Client applications are captured in the present setting by service queries. They are defined in a similar manner to service clauses and their semantics are, as expected, existential rather than universal.

Definition 18 (Query). *A* SOC-query *is a structure* (α, Q)*, also denoted*

$$\underset{\alpha}{\vdash\!\!-} Q$$

such that α *is an ARN and* Q *is a finite set of* α*-sentences referring to distinct requires-points of* α*. With respect to semantics,* $\mathcal{G}$ *satisfies* (α, Q) *if and only if there exists an interpretation of* α *satisfying all sentences in* Q*.*

Example 7. Figure 7 outlines the ARN of a possible client for the Journey Planner service. We specify the actual client through the service query

$$\underset{\text{Client}}{\vdash\!\!-} \left\{ @_{\mathsf{R}_1}\, \rho_1^{\mathrm{C}} \right\}$$

given by the MA-LTL-sentence $\Box\,(\mathsf{getRoute}_{\mathit{i}} \supset \Diamond\, \mathsf{route}_{\mathit{i}}')$.

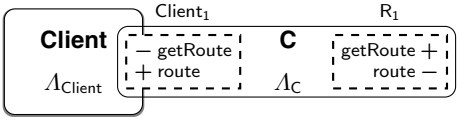


Fig. 7. The Client ARN

3.2 Resolution as Service Discovery and Binding

Service discovery represents, as in conventional LP, the search for a clause that could take the current goal one step closer to a possible solution. The solutions to service queries are defined in the same way as the solutions to first-order queries, but with ARN morphisms in the role of term substitutions.

Definition 19 (Solution). *A* solution *to* (α, Q) *consists of a morphism* $\theta\colon \alpha \to \alpha'$ *such that any interpretation of* α' *satisfies the* θ*-translation of any sentence in* Q*.*

The following result relates satisfiable queries and queries that have solutions, and may be regarded as a service-oriented correspondent of Herbrand's theorem.

Proposition 5. *A service query* (α, Q) *is satisfiable if and only if it admits a solution* $\theta\colon \alpha \to \alpha'$ *such that the category* $\mathsf{Mod}^{\underline{\mathrm{SOC}}}(\alpha')$ *is not empty.*

The procedure that ultimately decides whether or not a service can be bound to an application is *unification*.

Definition 20 (Unifier). *Let* $@_{x_1}\, \rho_1$ *and* $@_{x_2}\, \rho_2$ *be two sentences of ARNs* α_1 *and* α_2*, respectively. A* unifier *of* $@_{x_1}\, \rho_1$ *and* $@_{x_2}\, \rho_2$ *consists of a pair* $\langle \theta_1, \theta_2 \rangle$ *of morphisms* $\theta_1\colon \alpha_1 \to \alpha$ *and* $\theta_2\colon \alpha_2 \to \alpha$ *such that* $\theta_1^v(x_1) = \theta_2^v(x_2)$ *and*

$$\theta^{pt}_{2,x_2}(\rho_2) \models^{\text{MA-}\underline{\text{LTL}}} \theta^{pt}_{1,x_1}(\rho_1)\ .$$

In conventional LP the resolution process simplifies the current goal and at the same time, through unification, yields computed substitutions that could eventually deliver a solution to the initial query. This process is accurately reflected in the case of SOC by *service binding*.

Definition 21 (Resolution). *A service query* (α, Q) *is said to be derived by* resolution *from* (α_1, Q_1) *and* (P_2, α_2, R_2) *using the* computed morphism $\theta_1 : \alpha_1 \to \alpha$ *when*

$$\frac{\underset{\alpha_1}{\longmapsto} Q_1 \qquad P_2 \underset{\alpha_2}{\longleftarrow} R_2}{\underset{\alpha}{\longmapsto} \mathsf{Sen}^{\mathrm{SOC}}(\theta_1)(Q_1 \setminus \{R_1\}) \cup \mathsf{Sen}^{\mathrm{SOC}}(\theta_2)(R_2)} \ (\theta_1)$$

- *there exists a unifier* $\langle \theta_1, \theta_2 \rangle$ *of a sentence* $R_1 \in Q_1$ *and* P_2*, and*
- Q *is the set of sentences given by the translation along* θ_1 *and* θ_2 *of the sentences in* $Q_1 \setminus \{R_1\}$ *and* R_2*.*

Example 8. Let us consider the query and the clause detailed in the Ex. 7 and 6. One can easily see that the Client-sentence $@_{\mathsf{R}_1}\, \rho_1^{\mathrm{C}}$ and the JourneyPlanner-sentence $@_{\mathsf{JP}_1}\, \rho^{\mathrm{JP}}$ are unifiable. They admit the unifier $\langle \theta_1, \theta_2 \rangle$ given by

$$\mathrm{Client} \xrightarrow{\theta_1} \mathrm{Client} \| \mathrm{JourneyPlanner} \xleftarrow{\theta_2} \mathrm{JourneyPlanner}$$

- the ARN Client$\|$JourneyPlanner depicted in Fig.8,
- the ARN morphism θ_1 that maps the point R_1 into JP_1, the communication hyperedge C into CJP and the messages $\mathsf{getRoute}$ and route of M_{R_1} into $\mathsf{planJourney}$ and $\mathsf{directions}$, respectively (while preserving all the remaining elements of Client),
- the ARN inclusion morphism θ_2.

It follows that we can derive by resolution a new service query, defined by the network Client$\|$JourneyPlanner and the set of sentences $\left\{@_{\mathsf{R}_1}\, \rho_1^{\mathrm{JP}}, @_{\mathsf{R}_2}\, \rho_2^{\mathrm{JP}}\right\}$.

$$\frac{\underset{\mathrm{Client}}{\longmapsto} \left\{@_{\mathsf{R}_1}\, \rho_1^{\mathrm{C}}\right\} \qquad @_{\mathsf{JP}_1}\, \rho^{\mathrm{JP}} \underset{\mathrm{JourneyPlanner}}{\longleftarrow} \left\{@_{\mathsf{R}_1}\, \rho_1^{\mathrm{JP}}, @_{\mathsf{R}_2}\, \rho_2^{\mathrm{JP}}\right\}}{\underset{\mathrm{Client}\|\mathrm{JourneyPlanner}}{\longmapsto} \left\{@_{\mathsf{R}_1}\, \rho_1^{\mathrm{JP}}, @_{\mathsf{R}_2}\, \rho_2^{\mathrm{JP}}\right\}} \ (\theta_1)$$

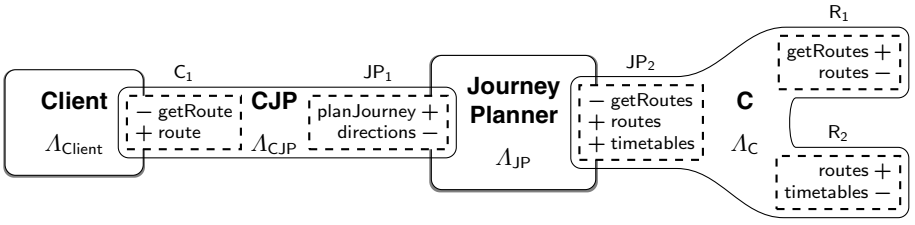


Fig. 8. The Client$\|$JourneyPlanner ARN

Computed Solutions. The process of service discovery and binding allows us to search for solutions to arbitrary queries. The search is triggered by a query (α, Q) and consists in the iterated application of resolution until the derived service query is empty, i.e. a query of the form $(\alpha', \emptyset)$. Whenever the search procedure successfully terminates we obtain a *computed solution* of the original query by sequentially composing the resulting computed morphisms.

The correctness of the search procedure relies on the correctness of resolution.

Proposition 6. *Let* (α, Q) *be a service query derived by resolution from* (α_1, Q_1) *and* (P_2, α_2, R_2) *using the computed morphism* θ_1. *If* (P_2, α_2, R_2) *is satisfiable then for any solution* θ *of* (α, Q), $\theta_1; \theta$ *is a solution of* (α_1, Q_1).

It is easy to see that any empty query admits a trivial solution, namely the solution given by the identity morphism of its underlying ARN. By applying Prop. 6 backwards, for each resolution step, we deduce that the composition of any terminating sequence of computed morphisms, and thus any computed solution, is a solution.

4 Conclusions

In this paper, we showed how the integration of declarative and operational semantics as provided by Logic Programming can be generalised to Service-Oriented Computing to offer an integrated semantics for the static and dynamic aspects of this paradigm, i.e. to provide, for the first time, an algebraic framework that accounts for the mechanisms through which service interfaces can be orchestrated and for those that allow applications to discover and bind to services. The analogy that we established is based on the identification of the binding of terms to variables in LP with the binding of orchestrations of services to requires-points of software applications in SOC. The answer to a service query – the request for external services – is obtained through resolution using the service clauses (orchestrated service interfaces) available from a repository. This departs from other works on the logic-programming semantics of services such as [15] that considered implementations of the service discovery and binding mechanisms using constraint logic programming.

The analogy is grounded on a declarative semantics of service clauses defined over a novel institution whose models are asynchronous networks of Muller automata (service orchestrations) and whose sentences are linear temporal logic sentences expressing properties that can be observed at given interaction points of a network. Other logics could have been used instead of linear temporal logic, more specifically any institution such that (*a*) the category of signatures is (finitely) co-complete; (*b*) there exist co-free models along any signature morphism; (*c*) the category of models of any signature has products; (*d*) any model homomorphism reflects the satisfaction of any sentence. These results encourage us to further develop a unifying framework for the foundations of LP that incorporates ideas from existing concrete variants of the phenomena, not necessarily restricted to relational or service-oriented programming. One possible course of action would be to isolate the principles of the LP paradigm in an institutional setting. This assumes institution-independent versions of LP concepts such as Herbrand model, clause, substitution, unifier and resolution, some of which have already been considered in the literature (e.g. [5]).

Acknowledgements. The work of the first author has been supported by a grant of the Romanian National Authority for Scientific Research, CNCS-UEFISCDI, project number PN-II-ID-PCE-2011-3-0439. The authors also wish to thank Fernando Orejas for suggesting the use of hypergraphs and Antónia Lopes for many useful discussions that led to the present form of this paper.

References

1. Alonso, G., Casati, F., Kuno, H.A., Machiraju, V.: Web Services – Concepts, Architectures and Applications. Springer (2004)
2. Benatallah, B., Casati, F., Toumani, F.: Representing, analysing and managing web service protocols. Data Knowl. Eng. 58(3), 327–357 (2006)
3. Brand, D., Zafiropulo, P.: On communicating finite-state machines. J. ACM 30(2), 323–342 (1983)
4. Bruni, R., Gadducci, F., Lluch-Lafuente, A.: A graph syntax for processes and services. In: Laneve, C., Su, J. (eds.) WS-FM 2009. LNCS, vol. 6194, pp. 46–60. Springer, Heidelberg (2010)
5. Diaconescu, R.: Herbrand theorems in arbitrary institutions. Inf. Process. Lett. 90(1), 29–37 (2004)
6. Diaconescu, R.: Institution-Independent Model Theory. Studies in Universal Logic. Birkhäuser (2008)
7. Ferrari, G.-L., Hirsch, D., Lanese, I., Montanari, U., Tuosto, E.: Synchronised hyperedge replacement as a model for service oriented computing. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.-P. (eds.) FMCO 2005. LNCS, vol. 4111, pp. 22–43. Springer, Heidelberg (2006)
8. Fiadeiro, J.L., Lopes, A.: An interface theory for service-oriented design. In: Giannakopoulou, D., Orejas, F. (eds.) FASE 2011. LNCS, vol. 6603, pp. 18–33. Springer, Heidelberg (2011)
9. Fiadeiro, J.L., Lopes, A.: A model for dynamic reconfiguration in service-oriented architectures. In: Software and Systems Modeling, pp. 1–19 (2012)
10. Fiadeiro, J.L., Lopes, A., Bocchi, L.: Algebraic semantics of service component modules. In: Fiadeiro, J.L., Schobbens, P.-Y. (eds.) WADT 2006. LNCS, vol. 4409, pp. 37–55. Springer, Heidelberg (2007)
11. Fiadeiro, J.L., Lopes, A., Bocchi, L.: An abstract model of service discovery and binding. Formal Asp. Comput. 23(4), 433–463 (2011)
12. Fiadeiro, J.L., Schmitt, V.: Structured co-spans: An algebra of interaction protocols. In: Mossakowski, T., Montanari, U., Haveraaen, M. (eds.) CALCO 2007. LNCS, vol. 4624, pp. 194–208. Springer, Heidelberg (2007)
13. Foster, I.T., Kesselman, C. (eds.): The Grid 2: Blueprint for a New Computing Infrastructure. Morgan Kaufmann (2004)
14. Goguen, J.A., Burstall, R.M.: Institutions: Abstract model theory for specification and programming. J. ACM 39(1), 95–146 (1992)
15. Kona, S., Bansal, A., Gupta, G.: Automatic composition of semantic web services. In: ICWS, pp. 150–158. IEEE Computer Society (2007)
16. Lloyd, J.W.: Foundations of Logic Programming, 2nd edn. Springer (1987)
17. Perrin, D., Éric Pin, J.: Infinite Words: Automata, Semigroups, Logic and Games. Pure and Applied Mathematics. Elsevier Science (2004)
18. Su, J., Bultan, T., Fu, X., Zhao, X.: Towards a theory of web service choreographies. In: Dumas, M., Heckel, R. (eds.) WS-FM 2007. LNCS, vol. 4937, pp. 1–16. Springer, Heidelberg (2008)
19. Tarlecki, A.: Quasi-varieties in abstract algebraic institutions. J. Comput. Syst. Sci. 33(3), 333–360 (1986)

Preface to CALCO-Tools

The CALCO-Tools workshops provide a forum to present, demonstrate, and discuss systems, prototypes, and tools developed specifically for the design, checking, execution, and verification of (co)algebraic specifications, but also tools targeting different application domains while making core or interesting use of (co)algebraic techniques.

The CALCO-Tools workshop series was initiated at the 2nd International Conference on Algebra and Coalgebra in Computer Science, CALCO 2007 in Bergen, Norway. It has formed an integral part of CALCO in the subsequent editions of the conference, which took place in Udine (2009) and Winchester (2011). Since 2011, CALCO-Tools has been taking place on the same dates as the CALCO main conference in order to emphasize and foster the mutual interaction between theory and tool development.

CALCO-Tools 2013 received seven submissions. Both the tool descriptions and the actual tools were carefully evaluated and discussed by the program committee, and found to be of high quality throughout, so that all seven submissions were accepted for the workshop. I expect the presentations of these tools will make for a stimulating and fruitful workshop.

I wish to thank the program committee members for their excellent and dedicated work both in the refereeing phase and in the discussion phase of the process, and the main conference chairs as well as the local organizers for their professional support.

June 2013 Lutz Schröder

Program Committee

Einar Broch Johnsen, University of Oslo, NO
Mark Hills, CWI Amsterdam, NL
Barbara König, University of Duisburg-Essen, DE
Dorel Lucanu, Alexandru Ioan Cuza University, Iasi, RO
Dominik Luecke, CWI Amsterdam, NL
Till Mossakowski, DFKI GmbH, Bremen, DE
Lutz Schröder, Friedrich-Alexander-Universität Erlangen-Nürnberg, DE (chair)
Alexandra Silva, Radboud University Nijmegen and CWI Amsterdam, NL

R. Heckel and S. Milius (Eds.): CALCO 2013, LNCS 8089, p. 314, 2013.

Checking Conservativity with HETS*

Mihai Codescu[1], Till Mossakowski[2,3], and Christian Maeder[2]

[1] University of Erlangen-Nürnberg, Germany
[2] DFKI GmbH Bremen, Germany
[3] SFB/TR 8 "Spatial Cognition", University of Bremen, Germany

1 Introduction

Conservative extension is an important notion in the theory of formal specification [8]. If we can implement a specification *SP*, we can implement any conservative extension of *SP* as well. Hence, a specification can be shown consistent by starting with a consistent specification and extending it using a number of conservative extension steps. This is important, because during a formal development, it is desirable to guarantee consistency of specifications as soon as possible. Checks for conservative extensions also arise in calculi for proofs in structured specifications [12,9]. Furthermore, consistency is a special case of conservativity: it is just conservativity over the empty specification. Moreover, using consistency, also *non-consequence* can be checked: an axiom does *not* follow from a specification if the specification augmented by the negation of the axiom is consistent. Finally, [3] puts forward the idea of simplifying the task of checking consistency of large theories by decomposing them with the help of an architectural specification [2]. In order to show that an architectural specification is consistent, it is necessary to show that a number of extensions are conservative (more precisely, the specifications of its generic units need to be conservative extensions of their argument specifications, and those of the non-generic units need to be consistent).

In this paper we present a (sound, but incomplete) algorithm for deciding conservativity of extensions of specifications in CASL [4] as part of the Heterogeneous Tool Set (HETS) [10], available at `http://hets.dfki.de`.

2 Conservative Extensions in CASL

CASL [4] extends many-sorted first-order logic with partial functions and subsorting. It also provides induction sentences, expressing the (free) generation of datatypes.

CASL signatures consist of a set S of sorts with a subsort relation $\leq$ between them together with families $\{PF_{w,s}\}_{w\in S^*,s\in S}$ of partial functions, $\{TF_{w,s}\}_{w\in S^*,s\in S}$ of total functions and $\{P_w\}_{w\in S^*}$ of predicate symbols. Signature morphisms consist of maps taking sort, function and predicate symbols respectively to a symbol of the same kind in the target signature, and they must preserve subsorting, typing of function and predicate symbols and totality of function symbols.

For a signature Σ, terms are formed starting with variables from a sorted set X using applications of function symbols to terms of appropriate sorts, while sentences are

* This work has been supported by the BMBF project SHIP.

R. Heckel and S. Milius (Eds.): CALCO 2013, LNCS 8089, pp. 315–321, 2013.

partial first-order formulas extended with *sort generation constraints* which are triples (S', F', σ') such that $\sigma' : \Sigma' \to \Sigma$ and S' and F' are respectively sort and function symbols of Σ'. Partial first-order formulas are translated along a signature morphism $\varphi : \Sigma \to \Sigma''$ by replacing symbols as prescribed by φ while sort generation constraints are translated by composing the morphism σ' in their third component with φ.

Models interpret sorts as sets such that subsorts are injected into supersorts, partial/total function symbols as partial/total functions and predicate symbols as relations. Note that sorts are assumed to be interpreted as non-empty sets, unless they are introduced using the keywords **esort** or **etype** (for datatypes).

The satisfaction relation is the expected one for partial first-order sentences. A sort generation constraint (S', F', σ') holds in a model M if the carriers of the reduct of M along σ' of the sorts in S' are generated by function symbols in F'.

A theory Γ is a pair $\langle \Sigma, \Gamma \rangle$ where Σ is a signature and Γ a set of sentences. A theory morphism $\sigma\colon \langle \Sigma, \Gamma \rangle \longrightarrow \langle \Sigma', \Gamma' \rangle$ is a signature morphism $\sigma\colon \Sigma \longrightarrow \Sigma'$ such that the sentences in Γ are mapped by σ to logical consequences of Γ'. A theory morphism $\sigma\colon \langle \Sigma, \Gamma \rangle \longrightarrow \langle \Sigma', \Gamma' \rangle$ is

- *conservative*, denoted $\mathbf{Cons}(\sigma)$, if each $\langle \Sigma, \Gamma \rangle$-model has a σ-expansion to a $\langle \Sigma', \Gamma' \rangle$-model;
- *monomorphic* ($\mathbf{Mono}(\sigma)$), if such an expansion exists uniquely up to isomorphism, and
- *definitional* ($\mathbf{Def}(\sigma)$), if each model has a unique such expansion.

We moreover write $\mathbf{DontKnow}(\sigma)$ and $\mathbf{NotCons}(\sigma)$ when the conservativity of σ can not be determined or does not hold, respectively. The following implications hold: $\mathbf{Def}(\sigma) \implies \mathbf{Mono}(\sigma) \implies \mathbf{Cons}(\sigma)$.

CASL specifications are built starting with basic specifications which are just theories. An extension of specifications is written: SP_1 **then** ... **then** SP_n, where at each step $i = 2, \ldots n$, SP_i must extend the signature Σ_{i-1} constructed at the previous step to a correct CASL signature Σ_i.

3 The Conservativity Checker

Unfortunately already for first-order logic, neither the check for conservative, nor monomorphic, nor definitional extension are recursively enumerable, which means that there cannot be a complete (recursively axiomatized) calculus for them.

Let us consider a specification extension SP_1 **then** SP_2. The main idea of the algorithm for checking conservativity of this extension is to separate the definitions of SP_2 into definitions of new sorts and datatypes and definitions of new functions and predicates. Each of the definitions is then analysed individually, and for each of them we obtain a consistency status. The possible answers are not conservative (**NotCons**), no result has been obtained (**DontKnow**), conservative (**Cons**), monomorphic (**Mono**), definitional (**Def**), and they are ordered by their strength as follows:

$$\mathbf{NotCons} < \mathbf{DontKnow} < \mathbf{Cons} < \mathbf{Mono} < \mathbf{Def}$$

The result of the analysis of the conservativity status of the extension is then obtained as the minimum of the results of the analysis of all its definitions w.r.t. the order given above. We denote the conservativity calculus for extensions of specifications thus defined by $\vdash SP_1$ **then** $SP_2 \rhd$ **Status**.

3.1 Checking Conservativity for Sorts and Datatypes

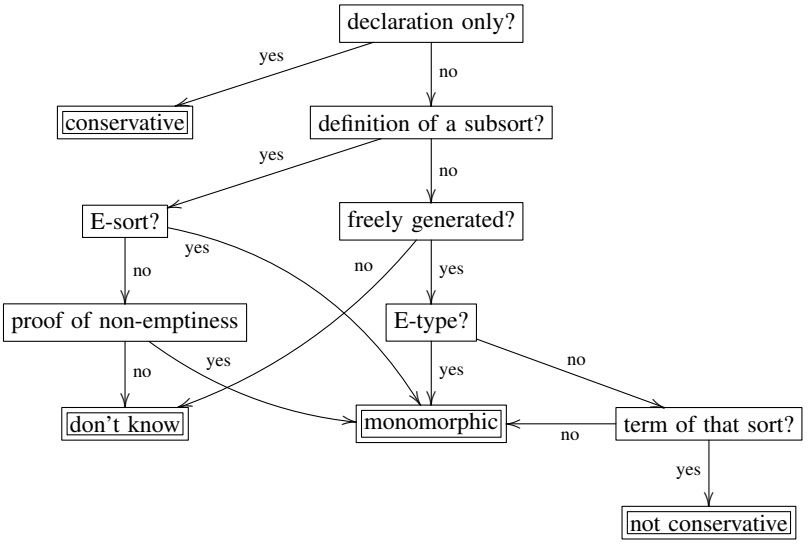


Fig. 1. Check for sorts and datatypes

If SP_2 contains no definitions of new sorts, then the result of analysis is just **Def** (the neutral element w.r.t the minimum operation). Otherwise, if a new sort or subsort has been declared without a definition, it does not depend on the old symbols and can be interpreted in any way, thus its status is **Cons**. If a sort is defined as a subsort of an existing sort with the help of a predicate Φ (as in $t = \{x : s.\Phi(x)\}$), then for a given model M of SP_1 the subsort t can be interpreted as any set isomorphic to the subset M_Φ. The result of analysis is thus **Mono**, provided that t has been declared as a **esort** or if the subset can be proven as non-empty. In the latter case a proof obligation is introduced and if it can not be discharged the status becomes **DontKnow**. If a sort t is defined as a free datatype, the status is **Mono**, provided that t has been declared as a **etype** or the specification ensures existence of a term of sort t. If this is not the case, the status becomes **NotCons**. This is summarised in Fig. 1.

3.2 Checking Conservativity for Functions and Predicates

Figure 2 shows the decision diagram for operation and predicate symbols. For each such symbol, all definitions involving the symbol are collected. Definitions are sentences of the following form

sentence	defined symbol	type of definition
$\forall x_1 : s_1, \ldots x_m : s_m \,.\, f(t_1, \ldots, t_n) = t$	f	function definition
$\forall x_1 : s_1, \ldots x_m : s_m \,.\, \neg def\, f(t_1, \ldots, t_n)$	f	function definition
$\forall x_1 : s_1, \ldots x_m : s_m \,.\, def\, f(t_1, \ldots, t_n) \Leftrightarrow \psi$	f	domain axiom
$\forall x_1 : s_1, \ldots x_m : s_m \,.\, p(t_1, \ldots, t_n) \Leftrightarrow \psi$	p	predicate definition

All sentences not associated to sorts are required to have one of these forms (any exception immediately leads to a **DontKnow**). So does any sentence which is a definition of an "old" symbol. The first check for a symbol is that for (syntactic) confluence (or proof of semantic confluence) of all its definitions. For total functions, we need to prove termination of the definitions in order to ensure totality. Sufficient completeness then implies that the function is uniquely defined; otherwise, we only know that the definition has at least one solution. For partial function symbols, the presence of a domain axiom is checked. If there is more than one, we end with **DontKnow**. If there is exactly one, we need to prove that the remaining definitions have a least fixed-point (taken for the function graphs) whose domain is captured by the domain axiom(s). If this proof succeeds, we can proceed as for total functions. If there are no domain axioms, we need to prove termination as well in order to proceed as with total functions. Otherwise, we still have conservativity.

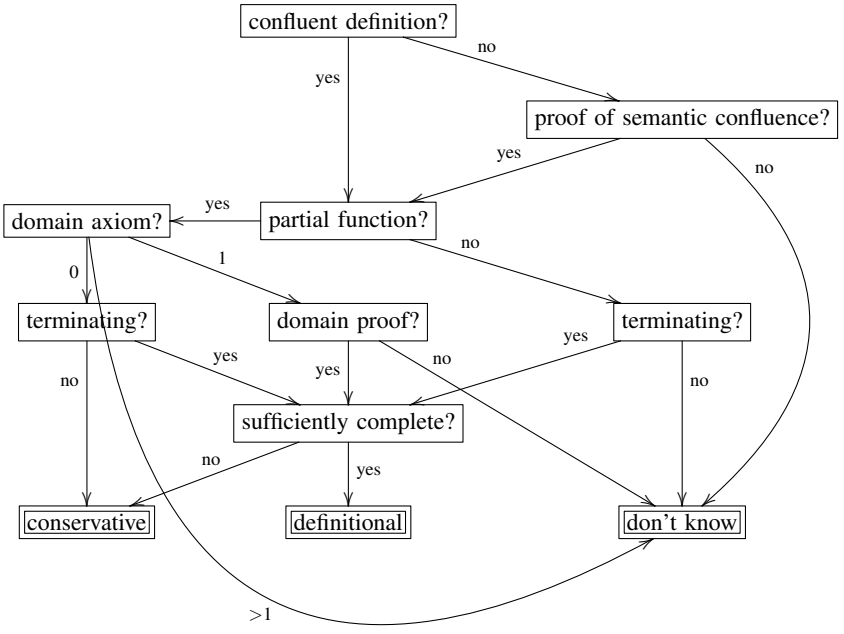


Fig. 2. Check for definitions of functions and predicates

The following result has been proved in [6].

Theorem 1 (Soundness). *If* $\vdash SP_1$ **then** $SP_2 \rhd$ **Status** *then* **Status**$(\iota : SP_1 \longrightarrow SP_2)$.

4 Examples

As an example, we present a simple but illustrative HETS library of natural numbers and operations on them. More examples, illustrating all branches of the diagrams, are available at `https://hets.dfki.de/Hets-lib/Conservativity/examples.het`.

spec NAT = **%mono**
free type $Nat ::= 0 \mid suc(Nat)$

spec NAT_COMP =
NAT **then %def**
free
{**pred** $\_\_<\_ : Nat \times Nat$
$\forall\, x, y : Nat$
$\bullet\ 0 < suc\ (x)$
$\bullet\ x < y \Rightarrow suc\ (x) < suc\ (y)$
}

spec POS =
NAT **then %mono**
sort $Pos = \{p : Nat \bullet \neg\, p = 0\}$

spec NAT_ADD =
NAT **then %def**
op $\_+\_ : Nat \times Nat \to Nat$
$\forall\, x, y : Nat$
$\bullet\ x + 0 = x$
$\bullet\ x + suc\ (y) = suc(x + y)$

spec NAT_PRED =
NAT **then %def**
op $pre : Nat \to ? Nat$
$\forall\, x : Nat$
$\bullet\ \neg\, def\ (pre(0))$
$\bullet\ pre(suc(x)) = x$

spec NAT_DIFF =
NAT_PRED **and** NAT_COMP **then %def**
op $\_-\_ : Nat \times Nat \to ? Nat$
$\forall\, x, y : Nat$
$\bullet\ def\ (x - y) \Leftrightarrow y < x \vee y = x$
$\bullet\ def\ (x - 0) \Rightarrow x - 0 = x$
$\bullet\ def\ (x - suc\ (y)) \Rightarrow x - suc\ (y) = pre(x - y)$

HETS analyses the specifications and produces a development graph [9]. Checking that NAT is monomorphic using HETS is done by right-clicking the node of NAT (its

yellow colour denotes that a conservativity proof obligation has been introduced) and selecting "Check conservativity", with the meaning that conservativity is checked over the empty signature. This also implies that NAT is consistent. HETS provides an interface for checking consistency of a theory directly, using a conservativity checker, by selecting "Check consistency" in the context menu of the node. This is illustrated in Fig. 3. In all other cases, the conservativity checker can be invoked by right-clicking the corresponding link (which is marked with **Mono?** or **Def?**) and selecting "Check conservativity".

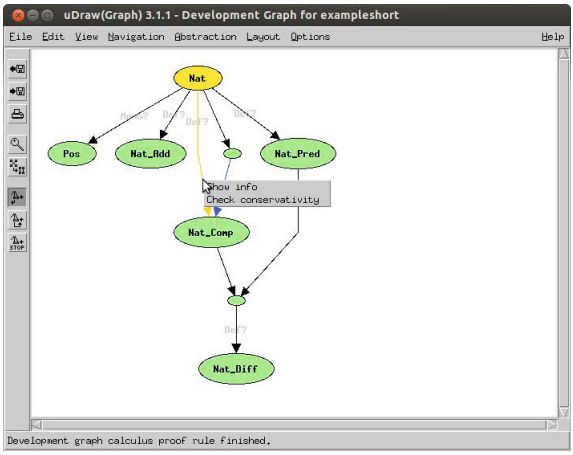


Fig. 3. Checking conservativity with HETS

HETS also uses conservativity to simplify the task of checking consistency of large theories. A node in a development graph is consistent if it has an incoming path with the origin in a node that has been proven to be consistent and such that all the morphisms contained in the path have been proven to be conservative.

We have run our conservativity checker for all the links in the CASL basic libraries [11]. The results are as follows:

don't know	790
conservative	455
conditionally conservative	49
monomorphic	25
conditionally monomorphic	11
definitional	322
sum	**1652**

The conditional variants indicate that some proof obligation has to be shown before conservativity (or monomorphicity) can be guaranteed.

5 Conclusion and Related Work

We have presented a tool (as part of the Heterogeneous Tool Set HETS) for checking conservativity of first-order specifications in CASL. In [5] we have shown the upper

ontology DOLCE to be consistent using HETS architectural specifications. This has involved 38 checks of consistency and conservativity, which all have been done with the HETS conservativity checker.

A tool with similar scope is the CASL consistency checker (CCC) [7]. However, it only provides a cumbersome-to-use calculus, whereas our tool uses a decision diagram giving automated results in most cases (proof obligations are generated only in a few cases). The CCC also has rules for structured specifications; in HETS these are realised as so-called development graphs rules [9].

Future work will integrate recent research on SMT solvers, in particular, the "big engine" approach with a generic base theory [1].

References

1. Beckert, B., Hoare, T., Hähnle, R., Smith, D.R., Green, C., Ranise, S., Tinelli, C., Ball, T., Rajamani, S.K.: Intelligent systems and formal methods in software engineering. IEEE Intelligent Systems 21(6), 71–81 (2006)
2. Bidoit, M., Sannella, D., Tarlecki, A.: Architectural specifications in CASL. Formal Aspects of Computing 13, 252–273 (2002)
3. Codescu, M., Mossakowski, T.: Refinement Trees: Calculi, Tools, and Applications. In: Corradini, A., Klin, B., Cîrstea, C. (eds.) CALCO 2011. LNCS, vol. 6859, pp. 145–160. Springer, Heidelberg (2011)
4. Mosses, P.D. (ed.): CASL Reference Manual. LNCS, vol. 2960. Springer, Heidelberg (2004)
5. Kutz, O., Mossakowski, T.: A modular consistency proof for Dolce. In: Burgard, W., Roth, D. (eds.) Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence and the Twenty-Third Innovative Applications of Artificial Intelligence Conference, pp. 227–234. AAAI Press, Menlo Park (2011)
6. Liu, M.: Konsistenz-Check von Casl-Spezifikationen. Master's thesis, University of Bremen (2008)
7. Lüth, C., Roggenbach, M., Schröder, L.: CCC – the CASL consistency checker. In: Fiadeiro, J.L., Mosses, P.D., Orejas, F. (eds.) WADT 2004. LNCS, vol. 3423, pp. 94–105. Springer, Heidelberg (2005)
8. Maibaum, T.S.E.: Conservative extensions, interpretations between theories and all that? In: Bidoit, M., Dauchet, M. (eds.) TAPSOFT 1997. LNCS, vol. 1214, pp. 40–66. Springer, Heidelberg (1997)
9. Mossakowski, T., Autexier, S., Hutter, D.: Development graphs – proof management for structured specifications. Journal of Logic and Algebraic Programming 67(1-2), 114–145 (2006)
10. Mossakowski, T., Maeder, C., Lüttich, K.: The Heterogeneous Tool Set, HETS. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 519–522. Springer, Heidelberg (2007)
11. Mossakowski, T., Hoffman, P., Autexier, S., Hutter, D., Mossakowski, E.: Part V. CASL Libraries. In: Mosses, P.D. (ed.) CASL Reference Manual. LNCS, vol. 2960, pp. 273–359. Springer, Heidelberg (2004), `http://www.informatik.uni-bremen.de/cofi/old/Notes/L-12/index.html`
12. Sannella, D., Tarlecki, A.: Foundations of Algebraic Specification and Formal Software Development. EATCS Monographs on theoretical computer science. Springer (2012)

The HI-Maude Tool

Muhammad Fadlisyah and Peter Csaba Ölveczky

University of Oslo

Abstract. In complex hybrid systems, different components may influence each others' *continuous* behaviors. HI-Maude is a rewriting-logic-based tool that supports an object-oriented modeling methodology in which it is sufficient to specify the continuous dynamics of *single* (physical component and physical interaction) objects in such interacting hybrid systems. HI-Maude supports simulation and model checking for a number of numerical approximations of the continuous behaviors, based on adaptations of the Euler and the Runga-Kutta methods.

1 Introduction

Many nontrivial hybrid systems consist of several components that may influence each others' *continuous* behaviors. The continuous behavior of such systems is typically very hard to define. Consider a system consisting of a cup of hot coffee in a room. The coffee will continuously become cooler *and* the room temperature will increase due to heat transfer from the coffee to the room. Although the continuous behaviors of the single components and the heat flow between them are well known and can be easily defined (see Fig. 1), it is very challenging to define the continuous behavior of the entire system "explicitly" in one shot, which is what current formal models of hybrid systems require.

In addition to making it practically impossible to define the continuous behaviors of nontrivial systems, such as the one discussed in Section 4, existing formalisms do not support an object-oriented specification methodology of continuously interacting hybrid systems, since the continuous behavior must be redefined for each new configuration of objects, e.g., if we have *three* cups of coffee in the room. Existing methodologies also do not support central object-oriented features such as dynamic creation and deletion of objects.

HI-Maude is a rewriting-logic-based formal tool for hybrid systems that supports an object-oriented modeling methodology in which both the physical components *and* their physical *interactions* are modeled explicitly. For example, heat flows from the coffee and the cup to the room through heat *convection*, and heat flows between the coffee and the cup through heat *conduction*. In HI-Maude, one can define the continuous dynamics of *single* physical component objects and single interaction objects. HI-Maude then computes the continuous dynamics of the entire system. This enables object-oriented modeling, since both the discrete and the continuous dynamics are defined at the class level, and since the dynamic creation/deletion of physical components is supported. For example, to add another cup of coffee, one could just add (possibly dynamically) a new coffee

R. Heckel and S. Milius (Eds.): CALCO 2013, LNCS 8089, pp. 322–327, 2013.

object, a new cup object, and three new interaction objects (for the convection between the new cup and the room and between the new coffee and the room, and for the conduction between the new coffee and the new cup) to the state.

To analyze hybrid systems—and HI-Maude targets complex systems whose continuous dynamics may be defined by differential equations that are not analytically solvable—HI-Maude uses an adaptation of different numerical methods (the Euler method and Runge-Kutta methods of different order) to give approximate solutions to coupled ordinary differential equations. These approximations are then used in HI-Maude simulation, reachability analysis, and linear temporal logic model checking. Since the numerical methods only approximate the real continuous behaviors, HI-Maude analyses are in general not sound and complete.

The HI-Maude tool, together with examples and documentation, is available at `http://folk.uio.no/mohamf/HI-Maude`. The paper [2] describes the adaptation of the numerical methods to the effort/flow modeling approach and compares the accuracy and execution times of the different methods in HI-Maude; the paper [3] presents HI-Maude and its semantic foundations in more detail; and [4] describes the case study summarized in Section 4.

2 Formal Modeling in HI-Maude

The HI-Maude modeling methodology is based on the the *effort/flow* method [5], in which a physical system is modeled as a network of *physical entities* and *physical interactions* between the entities. This approach is applicable to different systems. In mechanical translation systems, the pair of effort and flow variables are force and velocity; in mechanical rotation systems, torque and angular velocity; in electrical systems, voltage and current; in fluidic systems, pressure and volume flow rate; and in thermal systems, temperature and heat flow rate.

A *physical entity* is described by an *effort* value, a set of *attribute* values, and the entity's *continuous dynamics* (see Fig. 1, left). The effort variable represents a physical quantity, such as temperature, that evolves continuously, where the time derivative $\dot{e}$ of the effort e is a function of both the entity's attribute values *and* the flows of connected interactions (i.e., $\dot{e} = f(atts, \sum flows)$).

An *interaction* between two physical entities is described by a *flow*, a set of *attribute* values, and a *continuous dynamics*. The flow value's evolution over time is specified as function of the interaction's attributes and the efforts of the connected entities (i.e., $flow = g(atts, effort_1, effort_2)$). A *one-sided interaction* represents an interaction between a physical entity and its environment.

Figure 1 illustrates the effort/flow methodology on our coffee example, where the flow variable ($\dot{Q}$) denotes the heat flow rate. The effort variables T_R, T_C, and T_K of the room, the coffee, and the cup are their temperatures. The attributes denote parameters such as the mass and surface area of the coffee. An immersion heater that can be dynamically added to or removed from the system adds constant heat to the coffee. The system also exhibits discrete behaviors; e.g., the heater is turned off and on to keep the coffee temperature between 70° and 80°.

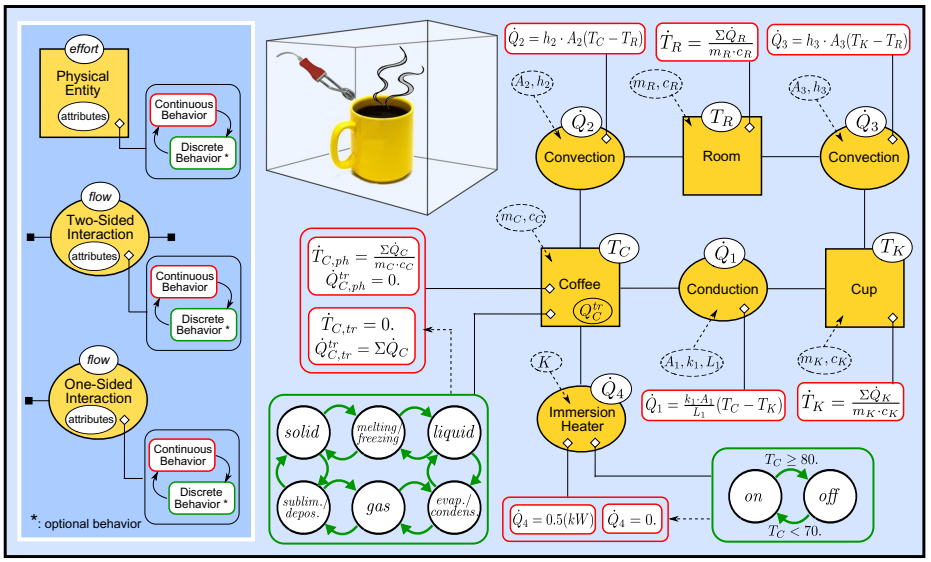


Fig. 1. Physical system components and their interaction in a simple thermal system

Since HI-Maude extends Maude [1], a *membership equational logic* theory (Σ, E), with Σ a signature[1] and E a set of *conditional equations*, specifies the system's state space as an algebraic data type. Instantaneous transitions are specified by conditional *rewrite rules* `crl` t `=>` t' `if` *cond*, where t and t' are two Σ-terms. A declaration `class` C `|` att_1 `:` s_1`, ...,` att_n `:` s_n declares a *class* C with attributes att_1 to att_n of sorts s_1 to s_n. An *object* of class C is represented as a term `<` O`:`C `|` att_1`:`val_1`,...,`att_n`:`val_n `>`, where O, of sort `Oid`, is the object's *identifier*, and where val_1 to val_n are the values of the attributes att_1 to att_n. The state is a term of sort `Configuration` denoting a *multiset* of objects and messages. A *subclass* inherits the attributes and rules of its superclasses.

In HI-Maude, physical entities and (one-sided and two-sided) interactions are defined as object instances of user-defined subclasses of the built-in classes

```
class PhysicalEntity | effort : Float .
class TwoSidedInteraction | flow : Float, entity1 : Oid, entity2 : Oid .
class OneSidedInteraction | flow : Float, entity : Oid .
```

The entity attributes denote the physical entities involved in the interaction. The user must define the time derivative `effortDyn(`$object, \sum \dot{Q}$`)` of the effort variable of *object* for each physical entity; the sum $\sum \dot{Q}$ of the flows to/from the entity is provided by the tool. For example, if $\dot{e} = f(atts, \sum flows)$, we define `effortDyn(<O : C |` $atts$ `>, X) =` f`(`$atts$`,X)`. We specify `flowDyn(`$object, e_1, e_2$`)` and `flowDyn(`$object, e_1$`)` to define the continuous dynamics of two-sided (resp. one-sided) *interaction objects*; the effort values e_1 and e_2 are given by HI-Maude.

[1] i.e., Σ is a set of declarations of *sorts*, *subsorts*, and *function symbols*.

Discrete transitions are modeled as rewrite rules, and `timeCanAdvance`(s) must be `false` in states s in which a rule must be applied.

To model the system in Fig. 1, we first define a class `ThermalEntity` (with attributes denoting the heat capacity and the mass of the entity), whose objects model thermal entities, such as the cup, the coffee, and the room:

```
class ThermalEntity | heatCap : Float, mass : Float .
subclass ThermalEntity < PhysicalEntity .
```

The `effort` attribute of the superclass denotes the temperature; its continuous dynamics is defined in the same way for each entity (e.g., $\dot{T}_R = \frac{\sum \dot{Q}_R}{m_R \cdot c_R}$):[2]

```
eq effortDyn(< O : ThermalEntity | mass : M, heatCap : C >, X) = X / (M * C) .
```

The heat flow by *convection* through the surface of the coffee or cup is given by the temperatures of the entities, the *area* of the surface (A), and the convection coefficient h. The definition of its dynamics is straight-forward from Fig. 1:

```
class Convection | area : Float, convCoeff : Float .
subclass Convection < TwoSidedInteraction .
eq flowDyn(< O : Convection | convCoeff : CC, area : A >, E1, E2) = CC * A * (E1 - E2).
```

The heater is a one-sided interaction (whose flow is 0 when the `status` is `off` and is 500 when the `status` is `on`; it also monitors the temperature of the coffee):

```
class Heater | status : OnOff, temp : Float.   subclass Heater < OneSidedInteraction.
eq flowDyn(< O : Heater | status : S >, E) = if S == on then 500.0 else 0.0 fi.
```

A rewrite rule turns off the heater when the temperature of the coffee has reached 80 degrees. `timeCanAdvance` forces the timely application of this rule:

```
crl < H : Heater | status : on, temp : T >  =>  < H : Heater | status : off >  if T >= 80.0.
ceq timeCanAdvance(< H : Heater | status : on, temp : T >) = false  if T >= 80.0 .
```

3 Formal Analysis in HI-Maude

HI-Maude provides a range of formal analyses, including: (i) simulating *one* behavior from a given initial state; (ii) checking whether a state matching a state pattern is reachable from the initial state (possibly within a given time interval); (iii) finding the shortest/longest time needed to reach an (un)desired state; and (iv) checking whether all possible behaviors from the initial state (possibly up to some duration) satisfy a linear temporal logic property. In all such analysis, HI-Maude advances time in small time increments and approximates the values of the continuous variables at each "visited" point in time. We have adapted the following numerical methods to our effort/flow framework: the *Euler*, the *Runge-Kutta 2nd order* (RK2), and the *Runge-Kutta 4th order* (RK4) methods.

[2] We follow the Maude convention that variables are written with (only) capital letters, and do not show the variable declarations.

In any analysis command, the user selects the numerical approximation technique and the time increment used in the approximation. For example, HI-Maude's hybrid *rewrite* command is used to simulate one behavior of the system from a given initial state *initState* up to duration *timeLimit*:

```
(hrew initState in time ~ timeLimit using numMethod stepsize stepSize .)
```

$\sim$ is either '`<=`' or '`<`'; *numMethod* $\in \{$`euler`, `rk2`, `rk4`$\}$; and *stepSize* is the time increment used in the approximations.

Using HI-Maude. The following module defines an initial state `c1` for the coffee example, consisting of one room, one heater, one coffee, and one cup object (each with temperature 20°), and the interaction objects in Fig. 1:

```
(homod COFFEE-SYSTEM is  including HYBRID-LIB .  including TIMED-MODEL-CHECKER .
...
eq c1 =
{< coffee : WaterEntity | effort : 20.0, mass : mC, heatCap : cC, phase : liquid, ... >
 < cup : ThermalEntity  | effort : 20.0, mass : mK, heatCap : cK, ... >
 < room : ThermalEntity | effort : 20.0, mass : mR, heatCap : cR, ... >
 < condCK : Conduction  | flow : 0.0, entity1 : coffee, entity2 : cup, thermCond : k1,
                          thickness : L1, area : A1, ... >
 < convCR : Convection  | flow : 0.0, entity1 : coffee, entity2 : room, area : A2,
                          convCoeff : h2 ... >
 < convKR : Convection  | flow : 0.0, entity1 : cup, entity2 : room, area : A3, ... >
 < immerHeater : Heater | flow : 0.0, entity : coffee, state : off, ... >} .
endhom)
```

HI-Maude simulation shows that after 5 minutes, the temperatures of the coffee, cup, and room are 79.47°, 74.43°, and 20.17°, respectively:

```
Maude> (hrew c1 in time <= 300 using euler stepsize 1.0 .)

Result ClockedSystem :
  {< coffee : WaterEntity | effort : 7.9470872797477682e+1, phase : liquid, ... >
   < cup : ThermalEntity  | effort : 7.4430660426346876e+1, ... >
   < room : ThermalEntity | effort : 2.0171582578947586e+1, ... > ... } in time 300
```

HI-Maude's *hybrid find earliest* command can then be used to find out how quickly the coffee temperature can reach 80°:

```
Maude> (hfind earliest
         c1 =>* {C:Configuration  < coffee : WaterEntity | effort : T:Float >}
          such that (T:Float >= 80.0) using euler stepsize 1.0 .)

Result: {< coffee : WaterEntity | effort : 8.00470...e+1, ... > ... } in time 277
```

Finally, we define a state proposition `temp-ok` to hold if the temperature of the coffee temperature is between 69.95° and 80.05°:

```
eq {REST  < coffee : WaterEntity | effort : T >} |= temp-ok = T >= 69.95 and T <= 80.05.
```

We can then model check (for all behaviors up to 30 minutes) that once an ok coffee temperature has been reached, it will remain in this interval:

```
Maude> (hmc c1 |=t [](temp-ok -> [] temp-ok) in time <= 1800 using euler stepsize 1.0.)

Result Bool :  true
```

4 Case Study: The Sauna World Championships

The winner of the Sauna World Championships is the contestant who can stay the longest in a $110°C$ sauna where half a liter of water is poured onto the sauna rocks every thirty seconds. The 2010 event ended in a tragedy when the two finalists collapsed with severe burn injuries after about six minutes; one of them died the next day. The cause of this tragedy is still under investigation.

Instead of experimenting with humans in different saunas to investigate the cause of this unsolved accident, one might use computers to analyze possible causes of the accident. However, that is a tricky task for reasons that include:

- The continuously changing temperature of the skin and of the "core" of the human body are different. Since problems may be caused by severe hyperthermia (body temperature above $40.6°C$) and second degree burn (skin temperature above $55°C$), we must take both values into account.
- The system also has instantaneous transitions (e.g., release of half a liter of water in the sauna) and nondeterministic behaviors (others leave the sauna).

We have defined a HI-Maude model of the human thermoregulatory system according to accepted physiological facts and models [4], where the body core and the skin are two main components. Heat flows between them through blood flows where the diameter of the blood vessels changes continuously. The main forms of heat exchange between skin and environment are by conduction/convection, radiation, and evaporation of sweat; between core and environment heat flows mainly through respiration. For the sauna, we model the heating rocks, their specific heat capacity, the pouring of water on the rocks, the heater, etc.

Our HI-Maude analyses (see [4]) show that even the average person should endure 12 minutes in the sauna before the onset of major injuries. Our results seem consistent with what we know about how long both professionals and amateurs can endure in the sauna. HI-Maude analysis showed that any of the following scenarios could explain the still unsolved tragedy that could cause major injuries to a five-time world champion in around 6 minutes:

- The initial temperature of the heating rocks is $250°C$.
- The sauna temperature of $210°C$.
- The initial humidity is very high (39 liters of water vapor instead of 10 liters).

References

1. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: All About Maude. LNCS, vol. 4350. Springer, Heidelberg (2007)
2. Fadlisyah, M., Ölveczky, P.C., Ábrahám, E.: Formal modeling and analysis of hybrid systems in rewriting logic using higher order numerical methods and discrete-event detection. In: Proc. CSSE 2011. IEEE (2011)
3. Fadlisyah, M., Ölveczky, P.C., Ábrahám, E.: Object-oriented formal modeling and analysis of interacting hybrid systems in HI-Maude. In: Barthe, G., Pardo, A., Schneider, G. (eds.) SEFM 2011. LNCS, vol. 7041, pp. 415–430. Springer, Heidelberg (2011)
4. Fadlisyah, M., Ölveczky, P.C., Ábrahám, E.: Formal modeling and analysis of human body exposure to extreme heat in HI-Maude. In: Durán, F. (ed.) WRLA 2012. LNCS, vol. 7571, pp. 139–161. Springer, Heidelberg (2012)
5. Wellstead, P.E.: Introduction to physical system modelling. Academic Press (1979)

Constructor-Based Inductive Theorem Prover

Daniel Găină, Min Zhang, Yuki Chiba, and Yasuhito Arimoto

Japan Advanced Institute of Science and Technology (JAIST),
Nomi, Ishikawa, Japan
{daniel,zhangmin,chiba,arimotoy}@jaist.ac.jp

Abstract. Constructor-based Theorem Prover (CITP) is a tool for proving inductive properties of software systems specified with constructor-based logics. CITP is equipped with a default proof strategy for the automated verification of Observational Transitional Systems (OTS), but the area of applications is not restricted to OTS. The proof strategy can be customised by the user, or the basic tactics can be applied step-by-step. The tool features are exhibited on concrete examples, showing how to perform verification with CITP.

1 Introduction

CITP is currently implemented in Maude [3] but its underlying logic is constructor-based order-sorted preorder algebra [8]. Unlike Maude ITP [9] or CIRC [10], the axioms of the specifications may include rewrite rules as well as equational and membership axioms. The denotational semantics of modules is restricted to reachable models w.r.t. given constructors [2]. It follows that the induction schemes are available not only for the specifications declared with an initial semantics but also for the specifications with a loose semantics [2,4]. In constructor-based logics the existence of initial models is guaranteed by the *sufficient completeness* property (see [5] for details). However, the soundness of the induction schemes is independent of sufficient completeness. Roughly speaking, the induction schemes are valid for the specifications with a loose semantics even if some of the non-constructor operations are *underspecified*. These semantic features increase the expressivity power of specifications and allow modeling the non-deterministic behavior of systems (see section 3).

A goal $\mathtt{SP} \vdash E$ consists of a specification $\mathtt{SP}$ and set of formulas E rather than a single formula like in Coq [1]. The proof rules $\dfrac{\mathtt{SP}_1 \vdash E_1 \ldots \mathtt{SP}_n \vdash E_n}{\mathtt{SP} \vdash E}$ of the specification calculus can be regarded, upside down, as basic tactics for decomposing problems. By applying a tactic to a goal $\mathtt{SP} \vdash \mathtt{E}$ we obtain a set of goals $\{\mathtt{SP_1} \vdash \mathtt{E_1}, \ldots, \mathtt{SP_n} \vdash \mathtt{E_n}\}$ if some preconditions are satisfied. If it is not the case, the goal remains unchanged. CITP is based on a methodology that takes into account that both the specification and the set of formulas of the initial goal are changing during verification. The tool is equipped with tactics that preserve the confluence and termination properties of the specifications in the proof process.

R. Heckel and S. Milius (Eds.): CALCO 2013, LNCS 8089, pp. 328–333, 2013.

CITP supports a simultaneous induction scheme which can be applied to a goal which consists of a specification and a set of formulas rather than a single formula. The tool can perform a complex *case analysis* automatically. CITP and many examples can be downloaded from `http://www.jaist.ac.jp/ danielmg/citp`.

2 Tactics

The basic tactics of CITP consist of the proof rules of the specification calculus defined in [4] and refined for applications in [6]. They are divided into two categories, *general basic tactics* and *specialized basic tactics*. General basic tactics can be applied to any specification. In contrast, specialized basic tactics are designed for the data types declared with an initial semantics such as `BOOL`, `SEQUENCE` and `NAT`.

General Basic Tactics. We present the general basic tactics which are sound for all specifications.

Simultaneous induction (`SI`) applies induction to a goal $\mathtt{SP} \vdash \mathtt{E}$ consisting of a specification `SP` and a set of formulas `E`. The *induction variables* are specified by the command `(set ind on` *VarSet* `.)`. Each *induction case* is given by a sort preserving mapping $\mathtt{CON} : \mathtt{VarSet} \rightarrow \mathtt{ConSet}$. This tactic can be applied by giving the command `(apply SI .)`. See [6] for details about `SI`.

Except `SI`, all basic tactics are designed for goals consisting of a specification and a single formula. However, if a tactic (different from `SI`) is applied to a goal of the form $\mathtt{SP} \vdash \{\mathtt{e}_1, \cdots, \mathtt{e}_n\}$, the goal is decomposed into a set of subgoals $\{\mathtt{SP} \vdash \mathtt{e}_1, \cdots, \mathtt{SP} \vdash \mathtt{e}_n\}$, and then the tactic is applied to each $\mathtt{SP} \vdash \mathtt{e}_i$.

Case analysis (`CA`) adds conditions to the specification of a goal from conditional equations whose left hand sides match subterms of the formula to prove. Conditional equations marked with a string starting with `"CA-"` are used for case analysis (see section 3). The technical details for finding conditional equations and subterms to apply `CA` are described in [6]. This tactic can be applied by giving the command `(apply CA .)`.

Theorem of constants (`TC`) instantiates variables appearing in the formula of the input goal by fresh constants. Such constants are automatically generated. Sort information of such constants is added to the specification. This tactic can be applied by giving the command `(apply TC .)`.

Implication (`IP`) adds the condition of a quantifier-free sentence of a goal to the specification of the goal, as assumption. This tactic can be applied by giving the command `(apply IP .)`.

Reduction is applied automatically by the system. Any goal of the form (1) $\mathtt{SP} \vdash \mathtt{eq\ t} = \mathtt{t'}$, (2) $\mathtt{SP} \vdash \mathtt{mb\ t} : \mathtt{s}$ and (3) $\mathtt{SP} \vdash \mathtt{rl\ t} \Rightarrow \mathtt{t'}$ is reduced to the empty goal set if (1) `t` and `t'` have the same normal form, (2) the sort of `t` is a subsort of `s`, and (3) `t` can be rewritten to `t'` by applying the rewrite rules of `SP`, respectively. Maude search engine is invoked in the third case.

Specialized Basic Tactics. The specialized basic tactics presented in this paper are sound for the specifications that protect booleans, sequences and

natural numbers. That is, there is neither junk data nor confusion introduced in BOOL, SEQUENCE and NAT.

Contradiction is applied automatically by the system. Any goal of the form (1) SP $\vdash$ eq t = t′, (2) SP $\vdash$ mb t : s and (3) SP $\vdash$ rl t $\Rightarrow$ t′ is reduced to the empty goal set if (1) true and false have the same normal form, (2) true and false have the same normal form, and (3) true can be rewritten to false by applying the rewrite rules of SP, respectively. This tactic is complementary to *Reduction* since it is applied when *Reduction* does not return an empty goal set.

Inconsistency is applied automatically by the system. Any goal of the form SP $\vdash$ ceq t_1 = t_2 if $C_1 \bigwedge$ b = not b $\bigwedge C_2$ is reduced to empty goal set. This strategy avoids non-termination processes during verification. If the specification of a goal contains two equations eq s^n0 <= t and eq t <= s^m0 such that $m \leq n$ then the goal is reduced automatically to empty goal set.

Case analysis for sequences (CS) is a specific pattern matching rule which splits a goal into subgoals, changing the formula to prove by considering occurrences of elements into sequences. CS leaves unchanged the specification of the initial goal. This tactic can be applied by giving the command (apply CS .). It is crucial for Example 2 of section 3.

Derived Tactics. Every application of a basic tactic is preceded by a reduction of the ground terms occurring in the formulas of the goals to their normal forms. The application order of the basic tactics is crucial for automating the proof process. For CITP the default order is as follows: SI CA CS TC IP. This proof strategy is designed to preserve confluence of the specifications modeling OTS. CITP allows users to customise the proof strategy depending on the problem to solve (see Example 1). CITP supports also the application of any derived tactics such as (apply SI CA .). The goals on the leaves of the proof trees are discharged by *Reduction*, *Contradiction* or *Inconsistency*.

3 Examples

In this section we present two examples of proofs assisted by CITP to show how to interact with the tool and demonstrate its efficiency.

Example 1. This is a simple example illustrating the complexity of the case analysis performed by the tool. The conditional equations used for case analysis are marked with a string starting with "CA-". The functions F and G over the natural numbers are defined in the module FUN as follows:

```
ceq F(X) = 5 if X <= 7 [metadata "CA-A"].
ceq F(X) = 1 if 8 <= X [metadata "CA-B"].
ceq G(Y) = 2 if Y <= 4 [metadata "CA-1"].
ceq G(Y) = 7 if 5 <= Y [metadata "CA-2"].
```

Suppose we want to prove (goal FUN |- eq 9 <= G(F(X:Nat))+ G(X:Nat) = true .). We set the tactic (set tactic TC CA .) and then the goal is discharged by giving the command (auto .). Theorem of constants is applied first. The new specification consists of FUN enhanced with the declaration of

the constant `op X#1 : -> Nat`. The formula to prove is `eq 9 <= G(F(X#1))+ G(X#1) = true`. Case analysis is performed w.r.t. patterns given by the innermost sub-terms of `G(F(X#1))+ G(X#1)` matched by the conditional equations above: `F(X#1)` and `G(X#1)`. There are four cases: (a) `X#1 <= 4`, (b) `5 <= X#1 <= 7` (c) `8 <= X#1 <= 4` (d) `8 <= X#1`. Note that CITP can identify and discharge goals with inconsistent specifications such as (c).

Maude ITP does not support the application of such a complex case analysis automatically. CIRC is equipped with a similar tactic for case analysis but it cannot identify goals with inconsistent specifications such as case (c).

Example 2. The following example is a simplified version of the alternating bit protocol. One agent puts repeatedly pairs $< \mathtt{bit}, \mathtt{d} >$ of bits and encoded data into a `channel` which automatically decodes the information. When the agent gets $< \mathtt{bit}', \mathtt{d}' >$ from the `channel` such that $\mathtt{bit} = \mathtt{bit}'$, it is a confirmation that the data sent for decoding was received. In this case, the agent stores the data received into a `list` structure, alternates the bit, and selects the next encoded data for sending. We assume that the channel is unreliable, meaning that the data in the channel may be lost, but not exchanged or damaged.

The data sent by the agent are indexed by natural numbers: $\mathtt{data}(0)$, ..., $\mathtt{data}(\mathtt{s}^n 0)$, where `op data : Nat -> Data` is the constructor for the data items. The sort `Bit` has two constructors `op t : -> Bit` and `op f : -> Bit` since the bits can have only two values. The function `op not_ : Bit -> Bit` alternates the bits. The `channel` consists of sequences of bits and data of the form $< \mathtt{bit}_0, \mathtt{data}(\mathtt{i}_0) >, < \mathtt{bit}_1, \mathtt{data}(\mathtt{i}_1) >, \ldots, < \mathtt{bit}_n, \mathtt{data}(\mathtt{i}_n) >$. The `list` of decoded data items received by the agent consists of sequences of form $\mathtt{data}(\mathtt{i}_0), \mathtt{data}(\mathtt{i}_1), \ldots, \mathtt{data}(\mathtt{i}_n)$.

In the OTS/CafeOBJ method the transitions between the states of the system are modelled with constructor operators. For our specification, `ABPS`, the constructors are the following ones:

Contructor	Meaning
`init : -> Sys`	Initial state
`rec : Sys -> Sys`	Agent receives pairs of bits & data
`send : Sys -> Sys`	Agent sends pairs of bits & encoded data
`drop : Sys -> Sys`	Dropping one element of queue

The structure of a state is abstracted by the following observers, each one returning observable information about the state:

Observer	Meaning
`channel : Sys -> Channel`	channel structure in a given state
`bit : Sys -> Bit`	Agent's bit
`next : Sys -> Nat`	Number of data sent by Agent
`list : Sys -> List`	Lists of data received by Agent

The `ABPS` specification is declared with a loose semantics. Two states are equal if they are undistinguishable by observers. This fact is formally described by means of equations. At the initial state the agent's `bit` is `f`, the `channel` is `empty`, the list is `nil`, and the index of the data to send is `0`. The values of the

bit after the execution of the constructor rec are described by the following equations:

```
ceq bit(rec(S)) = bit(S)     if channel(S) = empty [metadata "CA-b1"].
ceq bit(rec(S)) = bit(S)     if < B,D >,C := channel(S) ⋀ B = not bit(S)
    [metadata "CA-b2"].
ceq bit(rec(S)) = not bit(S) if < B,D >,C := channel(S) ⋀ B = bit(S)
    [metadata "CA-b3"].
```

Since elements from arbitrary positions of the channel may be dropped, the specification of loosing data is more subtle. We use "underspecified" operations to model the dropping action. For instance, the constructor drop is specified by two operations x y : Sys -> Channel and the following equations:

```
ceq channel(drop(S)) = x(S),y(S)   if x(S),< B,D >,y(S) := channel(S)
    [metadata "CA-d1"].
ceq channel(drop(S)) = channel(S) if match(channel(S),x(S),y(S))= false
    [metadata "CA-d2"].
eq bit(drop(S)) = bit(S).
eq next(drop(S)) = next(S) .
eq list(drop(S)) = list(S) .
```

There are no equations to define the values of x and y meaning that each ABPS-model has its own interpretation of x and y. The non-determinism consists in the choice among models rather than the choice within models.

We prove that if channel contains agent's bit then on the lower positions of the channel all bits are equal to the agent's bit.

```
(inv =def crl B2:Bit => bit(S:Sys) if
 C1:Channel,< B1:Bit,D1:Data >,C2:Channel,< B2:Bit,D2:Data >,C3:Channel :=
 channel(S:Sys) ⋀ B1:Bit = bit(S:Sys).)
```

Note that the formula to prove is formalized as a rewrite rule. As an equation the above invariant would cause non-termination during the verification. For the following proof, both equational and rewrite rules have the same denotational semantics, that of equality. We start our proof by adding the lemma crl true => false if not bit(S) => bit(S) to ABPS:

```
(th ABPSL is inc ABPS .
 var S : Sys .
 crl [lemma-inc]: true => false if not bit(S) => bit(S)[nonexec].
endth)
```

The proof is as follows:

```
(goal ABPSL |- crl B':Bit => bit(S:Sys) if
C1:Channel,< B1:Bit,D1:Data >,C2:Channel,< B2:Bit,D2:Data >,C3:Channel :=
channel(S:Sys) ⋀ B:Bit = bit(S:Sys).)

 (set ind on S:Sys .)                  --- S:Sys is marked for induction
 (apply SI .)                          --- induction is applied
  (auto .)                                    --- init
  (auto .)                                    --- send
  (auto .)                                    --- drop
  (init lemma-inc by S:Sys <- x#1 .) (auto .) --- rec
```

After induction is applied four subgoals are generated corresponding to the four constructors. The subgoals corresponding to `init`, `send`, and `drop` are discharged automatically by the command `(auto .)`. In order to discharge the subgoal corresponding to `rec`, we initialise `lemma-inc` with `S = x#1`, where `x#1` is the induction constant generated by the tool, and then we apply `(auto .)` to prove it automatically. Case analysis generates a subgoal with an inconsistent specification which is discharged by the `Contradiction` tactic.

4 Conclusion

CITP is based on a solid theoretical foundation (see [8,5,7,4]). The methodology supported by CITP was defined in [6] and it coherently combines the general proof rules (which are valid for all specifications) with the specialized rules of deduction (which are defined for specification declared with an initial semantics). Algebraic specification languages have standard libraries with predefined modules. In order to perform verification of complex software systems it is crucial to have tactics for the initial data types that are often used in practice such as booleans, sequences or natural numbers. The challenge is how to integrate these tactics with the ones for loose semantics and to push the boundaries of automation.

References

1. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions. Texts in Theoretical Computer Science. Springer (2004)
2. Bidoit, M., Hennicker, R.: Constructor-based observational logic. J. Log. Algebr. Program. 67(1-2), 3–51 (2006)
3. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: All About Maude. LNCS, vol. 4350. Springer, Heidelberg (2007)
4. Futatsugi, K., Găină, D., Ogata, K.: Principles of proof scores in CafeOBJ. Theor. Comput. Sci. 464, 90–112 (2012)
5. Găină, D., Futatsugi, K.: Initial Semantics in Logics with Constructors. J. Log. Comput. (2013), `http://dx.doi.org/10.1093/logcom/exs044`
6. Găină, D., Lucanu, D., Ogata, K., Futatsugi, K.: On Automation of OTS/CafeOBJ method (submitted, 2013)
7. Găină, D.: Interpolation in logics with constructors. Theor. Comput. Sci. 474, 46–59 (2013)
8. Găină, D., Futatsugi, K., Ogata, K.: Constructor-based logics. J. UCS 18(16), 2204–2233 (2012)
9. Hendrix, J.D.: Decision Procedures for Equationally Based Reasoning. Technical Report, UIUC (2008)
10. Lucanu, D., Goriac, E.-I., Caltais, G., Roşu, G.: CIRC: A Behavioral Verification Tool Based on Circular Coinduction. In: Kurz, A., Lenisa, M., Tarlecki, A. (eds.) CALCO 2009. LNCS, vol. 5728, pp. 433–442. Springer, Heidelberg (2009)

A Timed CTL Model Checker for Real-Time Maude

Daniela Lepri[1], Erika Ábrahám[2], and Peter Csaba Ölveczky[1]

[1] University of Oslo, Norway
[2] RWTH Aachen University, Germany

Abstract. This paper describes a recent timed CTL (TCTL) model checker for Real-Time Maude. Our model checker is sound and complete for large classes of systems for which there were previously no TCTL model checkers. Furthermore, since Real-Time Maude also provides a formal analysis back-end to a number of modeling languages, our model checker also equips such languages with a TCTL model checker for free.

1 Introduction

Real-Time Maude [10] extends the rewriting-logic-based Maude language and tool [4] to support the formal modeling and analysis of real-time systems. In contrast to most formal tools for real-time systems—whose specification formalisms are fairly restrictive to ensure that key properties are always decidable—Real-Time Maude emphasizes expressiveness and ease of specification. This has made it possible to successfully apply the tool to large state-of-the-art network protocols, scheduling algorithms, etc., that cannot be formalized using the most well known real-time tools such as RED, Kronos, Uppaal and TSMV (see [10]).

For the same reason, Real-Time Maude has also proved to be a suitable semantic framework and formal analysis back-end for a number of modeling languages. For example, Real-Time Maude analysis has been integrated into timed model transformation tools, the graphical modeling tool Ptolemy II, the OSATE tool environment for the avionics modeling standard AADL, and a number of other modeling languages (see, e.g., [8,2]). These languages cannot be formalized by timed automata because of, e.g., state variables whose values can grow beyond any bound and the need for unbounded data structures.

Until now, Real-Time Maude could only analyze reachability and *untimed* temporal logic properties. However, one is often interested in *timed* (or *metric*) properties, such as "the airbag will deploy *within 10 ms.* of a crash," instead of untimed ones, such as "the airbag will *eventually* deploy after a crash."

This paper describes a recent *timed CTL* (TCTL) model checker for Real-Time Maude. Although TCTL model checking is decidable for timed automata, very few timed temporal logic model checkers exist today, and then only for fairly restricted formalisms. The state-of-the-art timed-automata-based tool Uppaal [3] supports model checking for only a quite limited subset of *non-nested* TCTL properties. Kronos [12] and RED [11] implement TCTL model checkers for,

R. Heckel and S. Milius (Eds.): CALCO 2013, LNCS 8089, pp. 334–339, 2013.

respectively, timed automata and linear hybrid automata. While the above tools are based on dense time, TSMV [7] implements a RTCTL[1] model checker for timed Kripke structures for discrete time domains.

Of course, the TCTL model checking problem for the expressive Real-Time Maude formalism is in general undecidable. However, we have proved in [5,6] that, for a large class of systems that satisfy some easily checkable conditions, our TCTL model checker is indeed sound and complete also for dense time. An important consequence is that Real-Time Maude now provides, *for free*, sound and complete TCTL model checking to many of the modeling languages mentioned above. Such model checking has already been integrated into the Ptolemy II tool, and has been used to discover a previously unknown bug in a Ptolemy II model of a traffic light system.

The paper [6] introduces the semantic foundations of our model checker, focusing on the soundness and completeness issues (which is extended to the whole TCTL in [5]). The tool, with associated papers, is available at `http://folk.uio.no/leprid/TCTL-RTM/`.

2 The TCTL Model Checker

A Real-Time Maude specification contains declarations of *sorts*, *subsorts*, and *function symbols*, and a set of conditional equations, which specify the system's state space as an algebraic data type, and must contain a specification of a sort `Time` modeling the time domain (which may be dense or discrete). Labeled conditional *instantaneous rewrite rules*, written `crl [`l`] :` t `=>` t' `if` *cond*, specify the system's *instantaneous* (i.e., zero-time) local transitions. Time elapse is modeled by a set of *tick (rewrite) rules*, which typically have the form

`crl [`l`] : {`t`} => {`t'`} in time` x `if` x `<=` u `/\` *cond* `.`

where x is a new variable of sort `Time` that denotes the *duration* of the rewrite. Such tick rules are not directly executable, since many values are possible for the new variable x. Real-Time Maude therefore offers different *time sampling strategies* [9] to execute such tick rules. The *maximal* time sampling strategy advances time by the maximum possible time elapse u in each application of the tick rule. The *fixed-increment* strategy advances time by a given time value r.

Timed CTL (*TCTL*) [1] extends CTL by adding interval time constraints on the temporal operators. A TCTL formula is defined by:

$$\varphi ::= \quad true \quad | \quad p \quad | \quad \neg\varphi \quad | \quad \varphi \wedge \varphi \quad | \quad E\,\varphi\,U_I\,\varphi \quad | \quad A\,\varphi\,U_I\,\varphi$$

where p is a state proposition and I is a time interval. We can define the usual abbreviations, such as $E\ F_I\ \varphi$, $A\ F_I\ \varphi$, $E\ G_I\ \varphi$, and $A\ G_I\ \varphi$, in the standard way. Intuitively, $E\ \varphi_1\ U_I\ \varphi_2$ holds in a state s if a state s' is reachable from s in time $t \in I$, and φ_2 is satisfied in s', and all states from s to s' satisfy φ_1.

[1] RTCTL is a timed extension of CTL with integer-valued time constraints.

State propositions are terms of sort `Prop` and are specified by (possibly conditional) equations of the form `{`*statePattern*`} |=` *prop* `=` *b*, for *b* a term of sort `Bool`, which defines the state proposition *prop* to evaluate to *b* in all states matching the given pattern. The syntax of TCTL formulas is fairly intuitive, e.g. $E\ \texttt{p1}\ U_{\leq 5}\ (\neg\texttt{p2} \wedge A\ G\ (E\ F_{(7,8]}\ \texttt{p3}))$ is written

```
E p1 U[<= than 5)](not p2 and AG (EF[o 7, 8 c] p3))
```

where a time interval is a term of the kind `[`c_1 τ_1`,` τ_2 c_2`]`, where c_1 (respectively c_2) is `o` if the time interval is left (respectively right) open, and is `c` if it is closed.

Some care must be taken when defining the validity of TCTL formulae in Real-Time Maude. Does the formula $A\ F_{[1,2]}$ *true* (in all paths, a state where *true* holds must be reached in some time $1 \leq \tau \leq 2$) hold in the model with the single (tick) rule `{`$f(y)$`} => {`$f(y+x)$`} in time` x `if` x `<=` $3-y$ and initial state `{`$f(0)$`}`? The satisfaction of a formula can be defined under two different semantics: the *pointwise semantics* considers all possible paths (hence $A\ F_{[1,2]}$ *true* does *not* hold, since we have a path `{`$f(0)$`} => {`$f(3)$`} =>` $\ldots$); in the *continuous semantics*, all possible states "in-between" are taken into account when deciding the formula (and hence $A\ F_{[1,2]}$ *true* holds). See [5] for details.

Our explicit-state TCTL model checker is implemented in Maude and analyses a formula φ under the *pointwise semantics*. The model checker provides the user with two model checking commands:

`(mc-tctl` t `|=` φ `.)` *and* `(mc-tctl-gcd` t `|=` φ `.)`

for t the initial state and φ a TCTL formula. The first one performs the model checking on the model obtained by applying the user-selected time sampling strategy on the original model. The second command executes the *gcd* strategy, which advances time by $\bar{r}$ time units, where $\bar{r}$ is *half* of the greatest common divisor of all tick durations from t using the maximal time sampling strategy and all the non-zero, non-infinite time bounds in φ. The point of the *gcd* strategy is that *pointwise* model checking the *gcd* theory $\mathcal{R}^{gcd(t_0,r,\varphi)}$ provides a sound and complete model checking procedure for the *continuous* semantics, which is the more natural semantics, for large classes of systems [6,5]. In particular, we have

$$\mathcal{R}, L_P, t \models_c \varphi \quad \Longleftrightarrow \quad \mathcal{R}^{gcd(t_0,r,\varphi)}, L_P, t \models_p \varphi,$$

under the following reasonable assumptions:

- applying a tick rule does not change the valuation of a state proposition;
- instantaneous rewrite rules can only be applied after *maximal* tick steps or after applying an instantaneous rule;
- the state space reachable from the given initial state is finite.

At the moment, the model checker returns a "yes/no" answer and does not generate counterexamples/witnesses, in line with current TCTL model checkers, which at most provide counterexamples for very limited subsets of TCTL.

3 Performance Comparison

We compare our model checker with TSMV and RED on the bridge crossing benchmark, in which four persons must cross a bridge at night using a lamp. Only one lamp is available and at most two persons can cross at the same time. Each person walks with a different speed. When crossing together, they must walk at the speed of the slower one.

We model a state by a lamp and a multiset of terms `person(`τ`,`b`)`, where τ is the crossing time and b is `true` iff the person has crossed the bridge. We use the same values as in the TSMV analysis; p_1 crosses in time 5, p_2 in time 10, p_3 in time 20, and p_4 in time 25. We define the initial state `init(N)` to be parametric in the "scaling factor" `N`:[2]

```
eq init(N) = person(5 * N,false)   person(10 * N,false)
             person(20 * N,false)  person(25 * N,false)  lamp(false) .
```

The state proposition `safe` holds if all persons have crossed the bridge:

```
op safe : -> Prop .
eq {person(T:Time, false) S:System} |= safe = false .
eq {S:System} |= safe = true [owise] .
```

From the initial state, the shortest crossing time is 60. Therefore, under the *pointwise* semantics, using the maximal time sampling, from *any* possible state, it is possible to reach the safe state in time 85 (in the worst case, the slowest guy is alone on the other side with the lamp), which can be verified as follows:

```
Maude> (mc-tctl {init(1)} |= AG EF[<= than 85] safe .)
```

However, under the *continuous* semantics, 85 time units may not be enough to reach a safe state: if the slowest guy has just started to cross the bridge, we have to wait for time $50 - \epsilon$ for her to be back with the lamps, since she does not turn around on the middle of the bridge. Hence, under the continuous semantics, we can only guarantee that, from any state, a safe state can be reached in *less than* time 110. Indeed, the `mc-tctl-gcd` command for the formula above returns `false` for time bound 85 and `true` for time bound < 110.

Table 1 shows a comparison between our explicit-state model checker and the symbolic state-space-representation tools TSMV and RED.[3] The first 4 rows show a comparison for the formula `AG EF[<= than (85 * `n`)] safe`, with `110` instead of `85` for the continuous semantics, for n the scaling factor. The last 4 rows show a comparison for the same formulae, where `init+(`m`)` is `init(1)` with m additional `P2` persons.[4] TSMV should be compared to the "pointwise" results of Real-Time Maude, while RED should be compared to the "continuous"

[2] All specifications are available at `http://folk.uio.no/leprid/TCTL-RTM/`.

[3] UPPAAL does not support model checking the above TCTL formula.

[4] The analysis was performed on a 1.87 GHz Intel® Xeon® with 128GB of RAM.

Table 1. Execution times (in seconds) for the bridge crossing problem

Initial state	TSMV	Real-Time Maude (pointwise)	Real-Time Maude (continuous)	RED 7.0
`init(1)`	0.003	0.040	1.779	0.406
`init(10)`	0.125	0.036	1.871	0.408
`init(100)`	1.261	0.034	1.876	0.404
`init(1000)`	53.511	0.036	1.914	0.406
`init+(2)`	0.062	0.086	23.573	0.871
`init+(4)`	0.139	0.184	100.940	1.989
`init+(8)`	0.363	0.431	500.941	14.712
`init+(12)`	0.805	0.779	1658.725	70.316

ones.[5] It is not surprising that RED—optimized for timed automata, of which this problem is an instance—was the fastest in the comparison.

References

1. Alur, R., Courcoubetis, C., Dill, D.: Model-checking in dense real-time. Inf. Comput. 104, 2–34 (1993)
2. Bae, K., Ölveczky, P.C., Meseguer, J., Al-Nayeem, A.: The SynchAADL2Maude tool. In: de Lara, J., Zisman, A. (eds.) FASE 2012. LNCS, vol. 7212, pp. 59–62. Springer, Heidelberg (2012)
3. Behrmann, G., David, A., Larsen, K.G.: A tutorial on Uppaal. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 200–236. Springer, Heidelberg (2004)
4. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: All About Maude. LNCS, vol. 4350. Springer, Heidelberg (2007)
5. Lepri, D., Ölveczky, P.C., Ábrahám, E.: Sound and complete timed CTL model checking of timed Kripke structures and real-time rewrite theories (submitted for publication, 2013), `http://folk.uio.no/leprid/TCTL-RTM/`
6. Lepri, D., Ábrahám, E., Ölveczky, P.C.: Timed CTL model checking in Real-Time Maude. In: Durán, F. (ed.) WRLA 2012. LNCS, vol. 7571, pp. 182–200. Springer, Heidelberg (2012)
7. Markey, N., Schnoebelen, P.: TSMV: A symbolic model checker for quantitative analysis of systems. In: QEST 2004. IEEE Computer Society (2004)
8. Ölveczky, P.C.: Semantics, simulation, and formal analysis of modeling languages for embedded systems in Real-Time Maude. In: Agha, G., Danvy, O., Meseguer, J. (eds.) Formal Modeling: Actors, Open Systems, Biological Systems. LNCS, vol. 7000, pp. 368–402. Springer, Heidelberg (2011)
9. Ölveczky, P.C., Meseguer, J.: Semantics and pragmatics of Real-Time Maude. Higher-Order and Symbolic Computation 20(1-2), 161–196 (2007)

[5] The pointwise semantics of TSMV and Real-Time Maude coincide when the maximal time sampling strategy is selected, while the continuous semantics of RED and Real-Time Maude coincide.

10. Ölveczky, P.C., Meseguer, J.: The Real-Time Maude tool. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 332–336. Springer, Heidelberg (2008)
11. Wang, F.: Efficient verification of timed automata with BDD-like data structures. STTT 6(1), 77–97 (2004)
12. Yovine, S.: Kronos: A verification tool for real-time systems. Software Tools for Technology Transfer 1(1-2), 123–133 (1997)

Hybridisation at Work

Renato Neves[1], Alexandre Madeira[2], Manuel A. Martins[3], and Luís S. Barbosa[1]

[1] HASLab - INESC TEC & Univ. Minho
nevrenato@gmail.com, lsb@di.uminho.pt
[2] HASLab - INESC TEC & Dep. Mathematics, Univ. Aveiro & Critical Software S.A.
{madeira,martins}@ua.pt
[3] CIDMA - Dep. Mathematics, Univ. Aveiro

Abstract. This paper presents the encoding of the hybridisation method proposed in [MMDB11, DM13] into the HETS platform.

Keywords: Hybrid logics, institutions, reconfigurable systems.

1 Introduction and Purpose

Hybrid logics [Bla00] are a brand of modal logics that provides appropriate syntax for the possible worlds semantics through nominals. In particular, it adds to the modal description of transition structures the ability to refer to specific states. This paves the way to an expressive framework for specifying complex software able to evolve through different execution configurations. In a number of papers, starting with [MFMB11], the foundations and methodological aspects of such a framework for *reconfigurability* have been developed, leading to a two-stage method:

- *globally* the system's dynamics is represented by a transition structure described in a hybrid language, whose states correspond to possible configurations;
- *locally* each state is endowed with a structure modeling the respective configuration specification.

The logic used locally depends on the application requirements. Typical candidates are equational, partial algebra or first-order logic ($\mathcal{FOL}$), but one may equally resort to multivalued logics or even to hybrid logic itself equipping, in the last case, each state with another (local) transition system. Instead of fixing a particular hybrid logic, a systematic method to develop on top of each local logic, the characteristic features of hybrid logic was proposed. This process is called *hybridisation* and was characterised in [MMDB11, DM13], framed in the context of the theory of institutions [GB92] to achieve greater generality.

The *hybridisation* process abstracts away the syntactic and semantic details, that are independent of the very essence of the hybrid logic idea. This has a number of benefits, even if paying the price of a heavy notational burden. One is the focus on the essential, the theoretical development not being hindered by irrelevant details. Another one concerns its applicability to a wide number of concrete instances, since all of them could be regarded as combinations between concrete versions of hybrid logics with

R. Heckel and S. Milius (Eds.): CALCO 2013, LNCS 8089, pp. 340–345, 2013.

other logical systems. In this sense, the hybridisation method can be seen as a source of logics for the specification of *reconfigurable* systems [MFMB11].

Institutions provide a systematic way to relate logics and transport results from one to another, which means that a theorem prover for the latter can be used to reason about specifications written in the former. This is achieved through a special class of maps between institutions, referred to conservative comorphisms.

This paper reports on the implementation of the method introduced in [MMDB11] along two different directions. Firstly the general hybridisation method is incorporated in the HETS platform [MML07] — parsing and static analysis for the hybridisation of any base institution already supported in HETS being provided. Secondly, the comorphism $\mathcal{H}$CASL $\rightarrow$ CASL is implemented, offering effective tool support for proofs on a number of $\mathcal{H}$CASL-sub-institutions, namely $\mathcal{HFOL}$ and hybrid propositional logic[1]. This provides for free the proof support environment of a particularly well established logic. Naturally, the final goal is to have in HETS for free a comorphism from a hybridised logic $\mathcal{HI}$ to $\mathcal{FOL}$ given that a comorphism exists from the base logic $\mathcal{I}$ to $\mathcal{FOL}$.

HETS has been described as a "motherboard" where different "expansion cards" can be plugged. These pieces are individual logics (with their particular analysers and proof tools) as well as logic translations. To make them *compatible*, logics are formalised as institutions and translations as comorphisms. Therefore, the integration of the hybrid specifications on the HETS platform is legitimate, since all formal requirements (e.g., that institutions exist, that a comorphism can be defined, etc.) are already guaranteed by the hybridisation process itself.

The code for this extension to HETS, as well as a set of hybrid specification examples, is available from GITHUB (`https://github.com/nevrenato/Hets_Fork`). Additionally a ready–to–use HETS system is provided in a virtual machine available at SUGARSYNC (`https://www.sugarsync.com/pf/D7620475_67336482_6511440`).

2 Hybridisation as a Plug-in to HETS

2.1 Hybridisation of CASL

The hybridisation process was first incorporated into HETS through its direct application to what is the platform *lingua franca*: CASL[MHST03]. A comorphism from the outcome $\mathcal{H}$CASL to CASL was also defined. Thus, assisted proof support for $\mathcal{H}$CASL becomes available for free. $\mathcal{H}$CASL specifications add to the usual ones in CASL a declaration of nominals and modalities. Sentences include the typical *hybrid* machinery and quantification over nominals. Thus, the respective grammar is extended as follows:

CFor' = **HFor** | ... ;
HFor = @ *n CFor'* | < *m* > *CFor'* | [*m*] *CFor'* | *Here n* | ! *n CFor'* | ? *n CFor'*;

[1] $\mathcal{H}$CASL consists of the hybridisation of the institution CASL with the models restricted to those with sorts commonly realised in all the states and with common realisation of the quantified variables.

where n is a nominal, m a modality, and the last two alternatives the universal and existential quantifiers over nominals. The grammar above besides extending sentences with the typical *hybrid* machinery also includes the quantification over nominals. The latter makes possible to define complex operators such as the $\downarrow$ binder, or the notion of a rigid designator

Notice also the need to use the keyword *Here*, when having a nominal for a sentence. Such is needed so that the parser for CASL does not take nominals as a common proposition. In the following section a more sophisticated mechanism to prevent this kind of ambiguities is discussed.

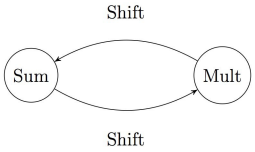


Fig. 1. Kripke frame for the swinging calculator

Figure 1 depicts a toy example of a calculator which commutes, through a modality $shift$, between two modes of operation: a binary operation is interpreted as arithmetic addition in mode Sum and multiplication in $Mult$. Its $\mathcal{H}$CASL encoding is reproduced below.

logic HYBRID
spec RECONFCALC =
HNAT
then modalities *Shift*
nominals *Sum*,*Mult*
ops $\_\#\_ : Nat \times Nat \to Nat$

%% global axioms
$\forall\ n, m, p : Nat \bullet n \# m = m \# n \wedge (n \# m) \# p = n \# (m \# p)$

%% axioms specific to Sum and Mult
$\forall\ n, m : Nat \bullet @Sum\ n \# 0 = n \wedge @Sum\ n \# suc(m) = suc(n \# m) \wedge @Mult\ n \# 0 = 0$
$\exists\ p, q : Nat \bullet @Mult\ n \# suc(m) = p \wedge @Sum\ n \# q = p \wedge @Mult\ n \# m = q$

%% axioms specific to the Kripke frame
$\bullet\ Here\ Sum \vee Here\ Mult$
$\bullet\ @Sum\ (< Shift > Here\ Mult \wedge [Shift]\ Here\ Mult)$
$\bullet\ @Mult\ (< Shift > Here\ Sum \wedge [Shift]\ Here\ Sum)$

%% lt relation definition, using # op
$\forall\ n, m, r : Nat \bullet n <= m \Rightarrow n \# r <= m \# r$

The encoding from $\mathcal{H}$CASL to CASL provides the expected proof support for this sort of hybrid specifications, *i.e.* the set of proof tools available for CASL is brought to $\mathcal{H}$CASL.

Below there are some examples of properties of the swinging calculator specification verified in this way.

Figure 2 registers the corresponding HETS session, showing the proof window, part of the model theory, and the specification graph.

- $@Sum\ n \# m >= n$ %(lemma2)%
- $@Mult\ (m = 0 \vee m = suc(0) \Rightarrow n \# m <= n)$ %(lemma6)%
- $@Sum\ [Shift]\ [Shift]\ Here\ Sum$ %(Cyclicity1)%

$\forall\ n, m, r : Nat$

- $n \# 0 = 0 \Rightarrow < Shift > n \# 0 = n$ %(StateExclusion)%
- $\exists\ p : Nat \bullet @Sum\ n \# n = p \Rightarrow @Mult\ n \# suc(suc(0)) = p$ %(DoubleDef)%
- $\exists\ p, q : Nat \bullet m <= suc(0) \Rightarrow @Sum\ n \# m = p \wedge @Mult\ n \# m = q \Rightarrow p >= q$ %(CasesSumBiggerMult)%

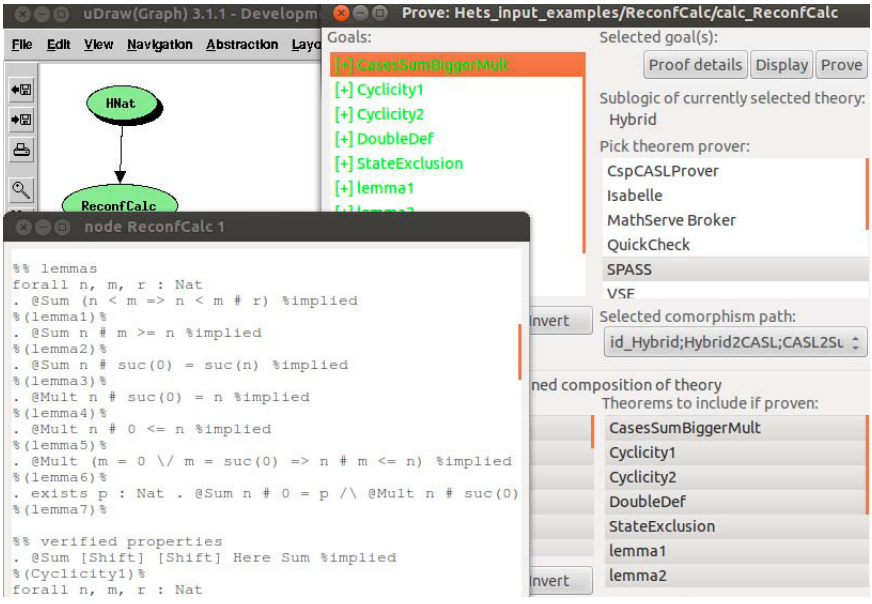


Fig. 2. A HETS session for the swinging calculator

2.2 Generic Hybridisation

The current integration of the hybridisation method into HETS offers the user hybridised versions of logics already "plugged in" HETS . Examples include propositional logic, CASL, COCASL, and any other logic previously hibridised. The implementation of the hybridisation framework can also be used to expand this list: given an identifier, a sentences' parser and an analyser, a new logic can be taken into the picture.

In each case the resulting grammar for writing *hybridised* specifications is the composition of a specification in the base logic, the declaration of nominals and modalities, and the sentences enriched with hybrid properties.

Note, finally, that *hybridisation* as described here may introduce ambiguities. In the example below, for instance, it is not clear which nominals belong to the base or the hybridised layer.

To clear out possible ambiguities resulting from double or multiple applications of the hybridisation method, formulas associated to the base layer are wrapped into curly brackets (i.e. they are transported to the hybridised level through an injection).
Follows an example specified in HETS using the *double hybridisation* of propositional logic:

logic HYBRIDIZE
spec GEOGRAPHY =
baselogic *Hybridize*
Basic_Spec{ **baselogic** *Propositional*
Basic_Spec { **props** *p* }
Nominals *Portugal*, *England*, *Canada*
Modalities *Car*
• @ *Portugal* <*Car*> *England* }
Nominals *Europe*, *America*
Modalities *Plane*
• *America* => { **not (** *Portugal* ∨ *England* **)** } ∧ *Europe* => { **not** *Canada* };
• @ *Europe* <*Plane*> **(** *America* ∧ { *Canada* } **)**

The specification above exemplifies a double hybridisation. It describes routes in a map linked by some means of transport (the modalities) between different places (identified by nominals). One level of hybridisation corresponds to countries; the second one to continents. Clearly, in this case nominals can be ordered respecting the order used to build an hierarchy of countries and continents. Note how this hierarchy is brought back into sentences. For instance the last one in the above specification states : *from Europe one can travel by plane to America; and, in particular, to Canada*.

3 Discussion

The *hybridisation* process and its implementation on HETS proved an effective and flexible way to prove properties of *hybrid* specifications and thus to support the design method in [MFMB11]. The implementation compares well with respect to dedicated provers for (specific) hybrid logics, although a systematic comparison is still being done. Typically, such tools, such as HTAB[HA09] or HYLORES[AH02], are faster to prove a formula with low complexity, but HETS achieves similar or even better performance in more complex ones. In some cases, formulas hard to deal with in HTAB are straightforward in HETS. A typical example is $A(\downarrow x\langle m\rangle\neg x)$.

Moreover, the genericity of the approach reported in this paper seems highly attractive in practice.

The results of [MMDB11, DM13] have yet a great potential to be explored on top of their integration in the HETS integration. The first reference shows that a comorphism from an arbitrary institution $\mathcal{I}$ to $\mathcal{FOL}$ gives rise to another comorphism from

its hybridisation $\mathcal{HI}$ to $\mathcal{FOL}$. Reference [DM13] refines this by characterising the conservativeness of such maps. Conservativeness is sometimes achieved not for the "free hybridisation" but for a restrict semantics of the hybridised institution satisfying a set of properties. Those restrictions are axiomatised on the $\mathcal{FOL}$ "side" as suitable presentations. The HETS rich support for $\mathcal{FOL}$ justifies the pertinence of the "hybridisation of comorphisms" method, since it extends tool support for a wide class of hybridised logics. Those includes not only hybrid equational logic and hybrid first-order logic (already supported by the comorphism presented above) but also hybridised modal logic and even hybridised hybrid propositional logic, among others.

Acknowledgements. This work is funded by ERDF - European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT, Portuguese Foundation for Science and Technology within project FCOMP-01-0124-FEDER-028923, and by *CIDMA-Centro de Investigação e Desenvolvimento em Matemática e Aplicações* of Universidade de Aveiro within the project FCOMP-01-0124-FEDER-022690.

References

[AH02] Areces, C., Heguiabehere, J.: HyLoRes 1.0: Direct resolution for hybrid logics. In: Voronkov, A. (ed.) CADE 2002. LNCS (LNAI), vol. 2392, p. 156. Springer, Heidelberg (2002)

[Bla00] Blackburn, P.: Representation, reasoning, and relational structures: A hybrid logic manifesto. Logic Journal of IGPL 8(3), 339–365 (2000)

[DM13] Diaconescu, R., Madeira, A.: Encoding hybridized institutions into first order logic (submitted, 2013)

[GB92] Goguen, J.A., Burstall, R.M.: Institutions: abstract model theory for specification and programming. J. ACM 39, 95–146 (1992)

[HA09] Hoffmann, G., Areces, C.: Htab: a terminating tableaux system for hybrid logic. Electr. Notes Theor. Comput. Sci. 231, 3–19 (2009)

[MFMB11] Madeira, A., Faria, J.M., Martins, M.A., Barbosa, L.S.: Hybrid specification of reactive systems: An institutional approach. In: Barthe, G., Pardo, A., Schneider, G. (eds.) SEFM 2011. LNCS, vol. 7041, pp. 269–285. Springer, Heidelberg (2011)

[MHST03] Mossakowski, T., Haxthausen, A., Sannella, D., Tarlecki, A.: CASL: The common algebraic specification language: Semantics and proof theory. Computing and Informatics 22, 285–321 (2003)

[MMDB11] Martins, M.A., Madeira, A., Diaconescu, R., Barbosa, L.S.: Hybridization of institutions. In: Corradini, A., Klin, B., Cîrstea, C. (eds.) CALCO 2011. LNCS, vol. 6859, pp. 283–297. Springer, Heidelberg (2011)

[MML07] Mossakowski, T., Maeder, C., Lüttich, K.: The heterogeneous tool set, HETS. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 519–522. Springer, Heidelberg (2007)

Penrose: Putting Compositionality to Work for Petri Net Reachability

Paweł Sobociński and Owen Stephens

ECS, University of Southampton, UK

Abstract. Recent work by the authors introduced a technique for reachability checking in Petri Nets, exploiting compositionality to increase performance for some well-known examples. We introduce a tool that uses this technique, `Penrose`, discuss some design details in its implementation, and identify potential future improvements.

1 Introduction

The famous example of Dining Philosophers has n philosophers around a dining table, contending for the use of shared forks, in order to eat. A Petri net[1] representation of three dining philosophers is given in Fig. 1. The graphical notation is non-standard, with "directed" places and undirected links[2].

Independent sets of transitions of a (1 bounded) Petri net can *fire* if the current marking contains tokens in the source place(s) and none in the target place(s). *Reachability* is the problem of determining if a particular *marking*—a set of places that contain a token—can be reached by firing transitions, starting from some initial marking. In this paper we introduce `Penrose`[3] a tool, written in Haskell, for solving reachability in Petri nets, via an algebraic approach.

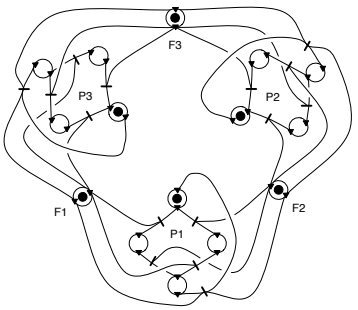


Fig. 1. Petri net representing a table of three dining philosophers

[1] Here we consider 1-bounded Petri nets, aka C/E nets or Elementary Net Systems.
[2] Places in the graphical presentation have separate in/out ports, and distinct transitions are marked with a stroke. For details see [12].
[3] Available for download: `http://users.ecs.soton.ac.uk/os1v07/Penrose_CALCO13`

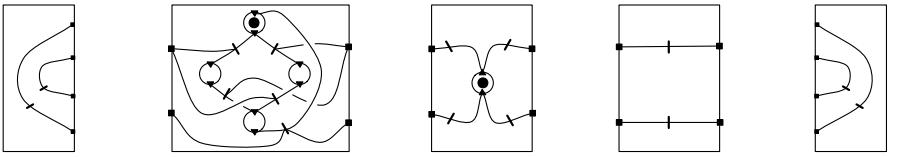
R. Heckel and S. Milius (Eds.): CALCO 2013, LNCS 8089, pp. 346–352, 2013.

Each Petri net determines a transition system with states the markings of the net and transitions that witness the simultaneous firing of an independent set of net transitions. For reachability, we consider the net's transition system as a non-deterministic finite automaton (NFA) over a unary alphabet: the initial and final states are, respectively, the initial and desired markings. Deciding reachability then coincides with emptiness of the NFA's language. This NFA is known as the *reachability graph* of a net; one algorithm is thus to construct the transition system, and determine if there is a path from the initial marking to the final one. State explosion makes this approach untenable: the number of markings (and thus, the statespace) is exponential in the number of places of the net.

1.1 A Local Approach: Penrose

Most standard approaches (e.g. [8,13]) to checking reachability are *monolithic* in that they consider a net as a whole. Penrose takes a different approach: decompose the net into small components (or take a decomposition as input), *locally* check reachability, and use the local information to reconstruct a global result. Our methodology is thus reminiscent of compositional model checking [5].

We use the algebra of *nets with boundaries* [11,3]. These extend Petri nets by adding left and right boundaries to a net, to which, transitions of the net can connect. They inherit the algebra of monoidal categories: composition can be "sequential", written ';', where two nets are synchronised, having their common boundary connected, or "tensor", written '$\otimes$', where two nets are placed "on top" of one another and considered as a single net. If N is a net with boundaries, we write $N : k \to l$ if N has a left boundary of size k and right boundary of size l.



$d_2 : 0 \to 4$ $ph : 2 \to 2$ $fk : 2 \to 2$ $i_2 : 2 \to 2$ $e_2 : 4 \to 0$

Fig. 2. Component nets with boundaries of Dining Philosophers

Using nets with boundaries, we can give a decomposition of the net in Fig. 1, in terms of 5 simple component nets, illustrated in Fig. 2: $PhRow_n$ is a row of n alternating philosophers and forks, and Ph_n a table of n dining philosophers:

$$PhRow_1 \stackrel{\text{def}}{=} ph \,;\, fk$$
$$PhRow_{k+1} \stackrel{\text{def}}{=} ph \,;\, fk \,;\, PhRow_k$$
$$Ph_n \stackrel{\text{def}}{=} d_2 \,;\, (i_2 \otimes PhRow_n) \,;\, e_2$$

The Ph_n construction "seals" the row of philosophers into a table, by wiring the last fork to the first philosopher, using d_2, i_2 and e_2.

Consider the deadlocked configuration, where all forks are picked up, but no philosophers are eating; this situation corresponds to a marking with fork places empty, and the "eating" places unmarked in each philosopher. A net with boundaries $N : k \to l$ determines an NFA in a similar fashion to a Petri net, however, the labels of the NFA's transitions are now important—they record the interaction of underlying sets of net transitions on the boundaries, as a pair of k-bit and l-bit binary strings[4]. The minimal DFA for $PhRow_2$ with the desired (local) marking is shown in Fig. 3 (the error state has been omitted, to increase readability.) The DFA illustrated is actually a fixed point: for $n \geq 2$, the minimised DFA of $PhRow_n$ is the same[5]. Similar observations have been made about dining philosophers modelled using the algebra of Span(Graph) [7].

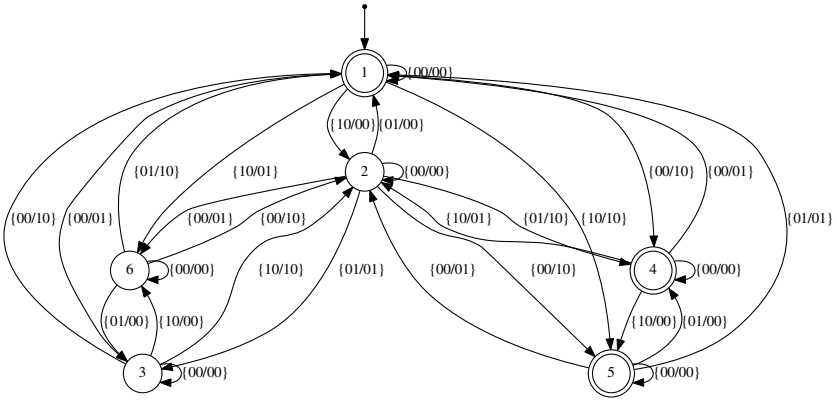


Fig. 3. Minimal DFA for $PhRow_2$

The automata representing the underlying net components are ';'- or '$\otimes$'-composed through modifications of the standard product construction.

The high-level algorithm of `Penrose` is:

1. Take as input a reachability problem, comprised of a Petri net, considered as a net with boundaries, and initial and final markings.
2. Transform the net into a *wiring decomposition* [12], or take one as input[6]. A wiring decomposition is a binary tree, with nodes composition operators, and leaves nets with boundaries (with local marking information).

[4] A '1' in the i^{th} position indicates the presence of an interaction with the i^{th} boundary port, on the corresponding side.

[5] In general, when we talk about *fixed points* we are referring to a situation in which there exists $k \in \mathbb{N}$ such that $\mathsf{Min}(N_k) = \mathsf{Min}(N_{k+1})$, where N_n is a recursively defined net built up from component nets with boundaries, and Min is some minimisation operation (DFA minimisation, quotienting by weak bisimilarity, etc.).

[6] Many *real* Petri nets have recursive specifications that are readily translated to the language of nets with boundaries.

3. Traverse the wiring decomposition, avoiding duplicate work via memoisation:
   - convert leaves (once per unique net) into corresponding NFAs and construct minimal DFAs to discard irrelevant local statespace.
   - at internal nodes, combine DFAs (once per unique pair) using either form of composition, and then minimise the resulting DFA.

DFA minimisation can be prohibitively expensive. On the other hand, it is a very coarse equivalence that allows us to prune the statespace aggressively, and in some examples allows us to reach a fixed point quickly, when using a finer equivalence would not suffice, as explained in Sec. 2. Properties of nets with boundaries [11,3] ensure correctness, see [12] for proofs.

2 Applicability and Performance

State explosion is common in model checking of concurrent systems; by *minimising* the automata of component nets, we obtain the minimal characterisation of their "protocol": how they must interact with the environment in order to reach a desired configuration. Prior to minimisation, we ϵ-close[7] the NFA, since only actions that interact with the net's boundaries affect its protocol. We then determinise and minimise, using Brzozowski's [4] algorithm. This algorithm is conceptually very simple: the NFA's transition relation is twice reversed and determinised (using the subset construction).

The performance of `Penrose` depends on two factors:

1. The *structure* of the input net: can we identify repeated "small" components?
2. The *semantics* of a decomposition: does the statespace explode or grow slowly as the net is reconstructed; do we reach a fixed point?

An initial investigation into structural issues has been carried out in [10]. Through memoisation, we can avoid duplicate work if repeated structure of a net is exposed and leads to a fixed point; for example, given the decomposition $PhRow_n$, no extra work is required to check reachability for $n \geq 3$, since a fixed point is reached at $n = 2$. This leads to performance that sometimes asymptotically outperforms monolithic approaches, see [12] for experiemental results. Characterising the underlying semantic issues is an open research question: *why* do Dining Philosophers reach a fixed point at $n = 2$?

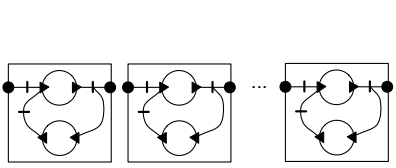
2.1 Minimisation and Fixed Points

Minimisation using Brzozowski's algorithm is potentially very expensive, since the subset construction is performed twice. Indeed, `Penrose` performs well only if small automata are minimised. The advantage to minimising w.r.t. language equivalence is that statespace is pruned aggressively—in particular, branching is discarded—and thus the likelihood of finding a fixed point[8] is greater.

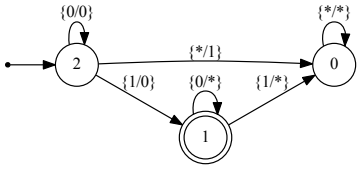
[7] ϵ-transitions are those with labels 0^*, indicating internal behaviour.

[8] `Penrose` finds fixed points via memoisation.

One alternative would be to quotient NFAs by (weak-) bisimilarity, obtaining smaller, equivalent NFAs without exponential blowup. However, on many examples bisimilarity is too fine an equivalence and fixed points do not exist because of branching, which is irrelevant for reachability. Indeed, quotienting by weak bisimilarity results in a fixed point only in deterministic variants[9] of the Dining Philosophers. We give another simple example of this phenomenon for the "replicators" of Fig. 4a. A replicator component can output an unbounded



(a) Chain of n Replicators



(b) Minimal DFA for n Replicators

number of tokens on the right after receiving a single token as input on the left. Consider a chain of n replicators, with the desired marking having a token only in the upper place of each; the chain's protocol is simple, and furthermore, is identical irrespective of n: after a single token has been received by the first replicator, it can be percolated through the chain with no interaction on the outermost boundary ports. This protocol is a fixed point reached at $n = 1$ and is the DFA shown in Fig. 4b. Quotienting by weak bisimilarity does not induce a fixed point. Therefore, in this example, the initial cost of determinisation pays off, whereas quotienting by bisimilarity is prohibitively expensive for large n.

3 Representing Transition Functions with BDDs

Recall that a net with boundaries N determines an NFA L, with labels binary strings of length $l = |\text{boundaries}(N)|$. A simple data structure for such an NFA is a set of pairs (s, f), where $s \in \mathsf{states}(L)$ and $f : \{0,1\}^l \to 2^{\mathsf{states}(L)}$, that is, a source state and function from labels to sets of (target) states.

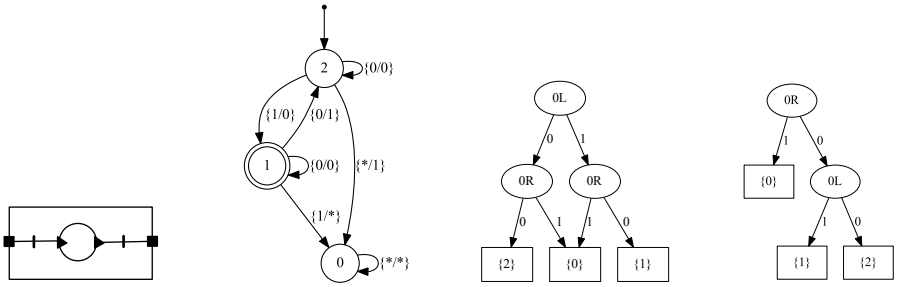
Reduced Ordered Binary Decision Diagrams (ROBDDs, or commonly just BDDs) are a compact representation of n-ary binary functions [1]. `Penrose` uses a generalisation of BDDs, Multi-Terminal BDDs [6], encoding functions with codomain the Boolean algebra of subsets $2^{\mathsf{places}(N)}$, rather than the Booleans.

As an example, consider the one place buffer net, B, and the reachability problem $(B, \langle absent \rangle, \langle present \rangle)$; we show B, the corresponding NFA and two alternative BDDs representing state 2's transition function, in Fig. 4.

BDDs are not necessarily minimal, with their size sensitive to variable ordering; for us, the lexicographical order[10] on boundary ports. Indeed, by reversing

[9] For example, philosophers that always take their left fork first.

[10] The ordering gives an interleaving left-right, top-bottom, on boundary ports (i, s) where $i \in \mathbb{N}, s \in \{L, R\}$ and $L < R$.



(c) Net B (d) NFA for B (e) $0L < 0R$ (f) $0R < 0L$

Fig. 4. One place buffer net with boundaries B, NFA for B with the initially empty, finally full marking and non-minimal and minimal BDDs for state 2

the variable ordering of Fig. 4e we obtain a smaller BDD, illustrated in Fig. 4f. For larger BDDs, the effect of reordering variables can be more dramatic. Computing optimal variable ordering is NP-Complete [2].

Penrose represents NFA transitions with a collection of BDDs, one for each state, but doing so loses potential sharing of common targets. For example, in Fig. 4d, the BDDs for state 2 and state 1 transitions will both have leaves for $\{0\}$. One possible solution is to use a data-structure similar to that of Minato et al. [9], where a single BDD is referenced by multiple "pointers" to nodes of the BDD graph, thereby retaining sharing.

4 Future Work

Penrose is under active development, currently supporting basic reachability-checking functionality outlined in this paper. We have obtained encouraging experimental results, sometimes asymptotically improving on monolithic approaches. Future work will investigate applying our decomposition technique to model checking problems other than reachability; optimising BDD representation, particularly w.r.t. the performance of the NFA to minimal DFA procedure; and further development of an algorithm for *automatic* decomposition of nets.

References

1. Andersen, H.R.: An Introduction to Binary Decision Diagrams. Technical Report October 1997, Technical University of Denmark (1997)
2. Bollig, B., Wegener, I.: Improving the Variable Ordering of OBDDs Is NP-complete. IEEE Transactions on Computers 45(9), 993–1002 (1996)
3. Bruni, R., Melgratti, H.C., Montanari, U., Sobociński, P.: Connector algebras for C/E and P/T nets' interactions. In: LMCS (to appear, 2013)
4. Brzozowski, J.: Canonical regular expressions and minimal state graphs for definite events. In: Mathematical Theory of Automata. MRI Symposia, vol. 12, pp. 529–561. Polytechnic Institute of Brooklyn, Polytechnic Press (1962)

5. Clarke, E.M., Long, D., McMillan, K.: Compositional model checking. In: LiCS 1989, pp. 352–362 (1989)
6. Fujita, M., McGeer, P.C., Yang, J.C.-Y.: Multi-Terminal Binary Decision Diagrams: An Efficient Data Structure for Matrix Representation. Formal Methods in System Design 10(2-3), 149–169 (1993)
7. Katis, P., Sabadini, N., Walters, R.: Compositional minimization in Span(Graph): Some examples. ENTCS 104C, 181–197 (2004)
8. McMillan, K.: A technique of a state space search based on unfolding. Form Method Syst. Des. 6(1), 45–65 (1995)
9. Minato, S.-I., Ishiura, N., Yajima, S.: Shared Binary Decision Diagram with Attributed Edges for Efficient Boolean Function Manipulation. In: Proc. DAC 1990, pp. 52–57. ACM Press (1990)
10. Rathke, J., Sobociński, P., Stephens, O.: Decomposing Petri nets. arXiv:1304.3121v1 (2013)
11. Sobociński, P.: Representations of Petri net interactions. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 554–568. Springer, Heidelberg (2010)
12. Sobociński, P., Stephens, O.: Reachability via compositionality in Petri nets. arXiv:1303.1399v1 (2013)
13. Starke, P.: Reachability analysis of Petri nets using symmetries. Systems Analysis Modelling Simulation 4/5, 292–303 (1991)

QStream: A Suite of Streams

Joost Winter[1,2,⋆]

[1] Centrum Wiskunde & Informatica (CWI)
[2] LIACS – Leiden University

Abstract. We present a simple tool in Haskell, QStream[1], implementing the technique of *coinductive counting* by making use of Haskell's built-in coinduction capabilities. We furthermore provide a number of useful tools for stream exploration, including a number of pretty print functions and integration with the Online Encyclopedia of Integer Sequences.

1 Introduction

It has been observed before, for example in [McI01] and [Hin11], that Haskell's built-in coinduction capabilities allow for easy and simple specifications of *streams* making use of variants of the coinductive stream calculus presented in e.g. [Rut05].

Borrowing some terminology from [McI01], the present paper can perhaps be considered as presenting another variation on this theme. Compared to [McI01], in which expressions for generating functions are given a coinductive semantics directly, our main focus lies on the connection with weighted automata, and systems of behavioural differential equations, using which rational and algebraic (or context-free) streams (or formal power series) can be chacaterized. We have opted for using a set of operators which is minimal but still expressive enough to be able to classify the complete classes of rational and algebraic streams.

Often, the specifications obtained this way turn out to be surprisingly elegant, although with a different flavour from the more familiar generating function expressions. For example, given a fixed integer k, the generating function for the stream of powers of k is $1/(1-kX)$, whereas the corresponding system of behavioural differential equations consists of the equations $o(x) = 1$ and $x' = kx$.

We have built a simple package, QStream, providing the necessary definitions required for such coinductive reasoning, as well as a usable and simple interface for stream exploration, including a number of pretty-printing functions for systems of streams, as well as an interface to the Online Encyclopedia of Integer Sequences[2].

Introductions to generating functions can be found, for example, in [Wil06] and [GKP94]. The idea of using coinductive techniques and weighted automata to describe combinatorial problems can be traced back to [Rut02]. For background material on the theory of rational and algebraic streams, we refer to [Rut08] and [BRW12], respectively.

⋆ Supported by the NWO project CoRE.

[1] The letter Q is intended to highlight the connection to automata theory.

[2] `http://oeis.org`

R. Heckel and S. Milius (Eds.): CALCO 2013, LNCS 8089, pp. 353–358, 2013.

The QStream package has been developed and tested using version 7.4.1 of The Glorious Glasgow Haskell Compilation System[3], and can be downloaded from `http://homepages.cwi.nl/~winter/qstream`.

Related Work: Existing tools aimed at stream calculus tend to fall into two distinct categories. The first category consists of tools, generally more 'heavy-weight' and with an emphasis on proving equality of streams: this group of tools includes CIRC [LGCR09] and Streambox[4].

The second category – in which computation of streams, rather than proving equality is the main aim – consists of more 'lightweight' implementations in Haskell. Earlier implementations in this category include those by McIlroy[5] and Hinze[6], which both have been a source of inspiration for QStream. Compared to these existing implementations, the present implementation attempts to provide a closer link with the underlying framework of systems of behavioural differential equations, as well as adding some useful interactivity by providing integration with the Online Encyclopedia of Integer Sequences.

2 A Suite of Streams

Elementary Coinductive Definitions: Haskell's built-in capabilities allow for easy coinductive specifications of streams. As a very elementary example, consider the specification `x = 1:2:3:x`. With this specification, it is directly possible to obtain initial segments of the stream thus defined as follows:

```
Prelude> take 10 x
[1,2,3,1,2,3,1,2,3,1]
```

Eventually periodic streams can, in general, be specified using specifications of this type. Although elementary, some important streams can already be defined now, such as the stream $0, 0, 0, \ldots$ defined by `zero = 0:zero`, and the stream $1, 1, 1, \ldots$, defined by `ones = 1:ones`.

In order to be able to produce more interesting classes of streams than the eventually periodic streams, we coinductively define a few basic operations on streams. Moreover, following McIlroy's example, we let streams be a `Num` type, enabling us to reap the fruits of type coercion, use the standard operators `+` and `*`, and furthermore directly inherit functions such as `sum`, `^`, etc. We also include a separate scalar product operator `*!`, which brings extra conceptual clarity as well as a tremendous performance boost.

The behavioural differential equations for sum, scalar product, and convolution product can be now represented directly in Haskell:

```
s  + t = o s + o t : d s  + d t       -- Or: (+) = zipWith (+)
k *! t =   k * o t :   k *! d t       -- Or: (*!) k = map ((*) k)
s  * t = o s * o t : d s  *   t + o s *! d t
```

[3] `http://www.haskell.org/ghc`
[4] `http://infinity.few.vu.nl/streambox/`
[5] `http://www.cs.dartmouth.edu/~doug/powser.html`
[6] `http://hackage.haskell.org/packages/archive/hinze-streams`

Here `o` and `d`, standing for *output* and *derivative* respectively, are simply defined as synonyms for `head` and `tail`.

Rational and Algebraic Streams: As a first example of a rational stream, consider the definition:

```
fibs = 0 : 1 : fibs + d fibs
```

This definition corresponds to the following system of behavioural differential equations

$$o(x) = 0 \qquad o(y) = 1 \qquad x' = y \qquad y' = x + y$$

and yields the Fibonacci sequence $0, 1, 1, 2, 3, 5, 8, \ldots$

Making use of the existing `oeis` package on Hackage, our module `QStream.IO` provides a function `info`, which takes a stream of integers as argument, looks up an initial part of this stream in the Online Encyclopedia of Integer Sequences (`http://oeis.org`), and then displays its description and identifier.

```
*QStream> info fibs
Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) =
1. (A000045)
```

For two more rational streams, consider:

```
dups = 1 : 2 *! dups
hypercube = 1 : 2 *! (hypercube + dups)
```

Here `dups` consists of the powers of 2: (1,2,4,8,...). The nth element of `hypercube`, on the other hand, is equal to the number of edges in a n-dimensional hypercube.

A celebrated example of an algebraic stream is the specification

```
cats = 1 : cats ^ 2
```

corresponding to the system of behavioural differential equations

$$o(x) = 1 \qquad x' = x^2$$

and yielding the stream of Catalan numbers $1, 1, 2, 5, 14, 42, 132, 429, 1430, \ldots$

Systems of Streams: It is also possible to define complete systems of streams at once. As an example, the following stream systems yield, respectively, diagonal rows from Pascal's triangle, and the Stirling numbers of the 2nd kind:

```
pascal n = 1 : sum [ pascal i | i <- [1..n] ] -- A007318
stirling2 n = 1 : sum [ i *! stirling2 i | i <- [1..n] ] -- A008277
```

These Haskell specifications are in direct correspondence to the systems of behavioural differential equations

$$o(p_n) = 1 \qquad p'_n = \sum_{i=1}^{n} p_i \quad (n \in \mathbb{N}) \qquad \text{giving: } [\![p_n]\!](k) = \binom{n+k}{k}$$

$$o(s_n) = 1 \qquad s'_n = \sum_{i=1}^{n} i s_i \quad (n \in \mathbb{N}) \qquad \text{giving: } [\![s_n]\!](k) = \frac{1}{k!} \sum_{j=0}^{k} (-1)^{k-j} \binom{k}{j} j^{n+k}$$

which can easily be derived from the familiar recurrence relations for these sequences.

For these types of stream systems, again a few helper functions are provided: `rangeinfo` gives the description and OEIS numbers of a stream system for a provided range of values:

```
*QStream> rangeinfo stirling2 [2..4]
2: 2^n - 1. (Sometimes called Mersenne numbers, although that
name is usually reserved for A001348.) (A000225)
3: Stirling numbers of second kind S(n,3). (A000392)
4: Stirling numbers of the second kind, S(n,4). (A000453)
```

Another pair of helper functions, `maketable` and `oeistable`, generates a table providing an initial part of a function from natural numbers to streams, with `oeistable` presenting the OEIS ids in addition:

```
*QStream> oeistable pascal 12 5
1:  1 1  1  1  1   1   1   1   1   1    1    1   (A000012)
2:  1 2  3  4  5   6   7   8   9  10   11   12   (A000027)
3:  1 3  6 10 15  21  28  36  45  55   66   78   (A000217)
4:  1 4 10 20 35  56  84 120 165 220  286  364   (A000292)
5:  1 5 15 35 70 126 210 330 495 715 1001 1365   (A000332)
```

Sometimes, systems of behavioural differential equations, and the corresponding specifications in Haskell, are much simpler in form than the sometimes more familiar explicit formulas for these sequences. For example, the number of m-ary search trees on n keys is equal to the nth element of the stream `searchtrees m`, specified by:

```
searchtrees m = take (m - 1) ones ++ searchtrees m ^ m
```

This equation can easily be derived from the generating function specification

$$A(X) = \sum_{j=0}^{m-2} X^j + X^{m-1} A^m(X),$$

found in e.g. [FD97], where a corresponding explicit formula (omitted here due to space constraints) is also provided.

Building a Catalog of Streams: In [Plo92], generating functions for 1031 different integer sequences have been identified using gfun, a Maple package. With QStream, we have so far found behavioural differential equations for over 100 of the generating functions presented there [Plo92]: this small catalog can be found in the module `QStream.Plouffe`.

3 Conclusions and Future Work

So far, QStream has been, at least for its author, a useful tool in exploration of classes of streams and systems of behavioural differential equations. The underlying theoretical framework links up beautifully with Haskell, and typical Haskell features such as lazy evaluation. Even merely experimenting around a bit with coinductive specifications often yields interesting sequences; as well as elegant specifications for these sequences.

However, when parameterized systems such as `pascal` are involved, computation of streams turns out to be awkwardly slow. Possible tactics to address this issue include memoization, and direct modelling of weighted automata using linear combinations of weighted states. As a first step in the second direction, the module `QStream.Fast` hard-codes weighted automata for a relatively wide class of streams (including the `pascal` and `stirling2` stream systems). This approach, albeit ad hoc, already yields a huge speed up, resulting in much more reasonable computation times. Further work here should include a more modular approach, in which data types representing weighted automata are introduced.

Looking up streams on OEIS can be a rather slow process: somehow, especially looking up basic sequences (such as `ones` or the natural numbers) often is inexplicably slow. Although this issue is mostly out of our control, to remedy this, we might think for example of building a local database of OEIS entries.

As a final remark, we note that there should be a number of easy generalizations of this work: for example by moving from integers to rationals, or from streams to formal power series over noncommuting variables (or, equivalently, weighted languages). However, in neither of these cases would we be able to make any good use of the OEIS, which focuses on integer sequences.

Acknowledgements. The author would like to thank Marcello Bonsangue and Jan Rutten for their comments and constructive criticism, as well as to the anonymous reviewers for their comments and suggestions.

References

[BRW12] Bonsangue, M.M., Rutten, J.J.M.M., Winter, J.: Defining context-free power series coalgebraically. In: Pattinson, D., Schröder, L. (eds.) CMCS 2012. LNCS, vol. 7399, pp. 20–39. Springer, Heidelberg (2012)

[FD97] Fill, J.A., Dobrow, R.P.: The number of m-ary search trees on n keys. Combinatorics, Probability & Computing 6(4), 435–453 (1997)

[GKP94] Graham, R.L., Knuth, D.E., Patashnik, O.: Concrete Mathematics: A Foundation for Computer Science, 2nd edn. Addison-Wesley Longman Publishing Co., Inc., Boston (1994)

[Hin11] Hinze, R.: Concrete stream calculus—an extended study. JFP 20(5-6), 463–535 (2011)

[McI01] Douglas McIlroy, M.: The music of streams. Inf. Process. Lett. 77(2-4), 189–195 (2001)

[LGCR09] Lucanu, D., Goriac, E.-I., Caltais, G., Roşu, G.: CIRC: A Behavioral Verification Tool Based on Circular Coinduction. In: Kurz, A., Lenisa, M., Tarlecki, A. (eds.) CALCO 2009. LNCS, vol. 5728, pp. 433–442. Springer, Heidelberg (2009)

[Plo92] Plouffe, S.: Approximations de séries génératrices et quelques conjectures. Master's thesis, Université du Québec à Montréal (1992)

[Rut02] Rutten, J.J.M.M.: Coinductive counting: bisimulation in enumerative combinatorics. Electr. Notes Theor. Comput. Sci. 65(1), 286–304 (2002)

[Rut05] Rutten, J.J.M.M.: A coinductive calculus of streams. Mathematical Structures in Computer Science 15(1), 93–147 (2005)

[Rut08] Rutten, J.J.M.M.: Rational streams coalgebraically. Logical Methods in Computer Science 4(3) (2008)

[Wil06] Wilf, H.S.: Generatingfunctionology. A. K. Peters, Ltd., Natick (2006)

Author Index

Ábrahám, Erika 334
Aceto, Luca 36
Arimoto, Yasuhito 328

Balan, Adriana 51
Barbosa, Luís S. 340
Bauer, Andrej 1
Bílková, Marta 66
Bojańczyk, Mikołaj 17
Bonchi, Filippo 80
Bonsangue, Marcello M. 95

Chiba, Yuki 328
Ciancia, Vincenzo 110
Codescu, Mihai 315

Dostál, Matěj 66

Enqvist, Sebastian 126

Fadlisyah, Muhammad 322
Fiadeiro, José Luiz 299
Furber, Robert 141

Găină, Daniel 328
Ghani, Neil 19
Goncharov, Sergey 158
Goriac, Eugen-Ioan 36
Gorín, Daniel 253

Hansen, Helle Hvid 95
Hirschowitz, Tom 175
Hölzl, Johannes 236

Ingolfsdottir, Anna 36

Jacobs, Bart 141

Kurz, Alexander 51, 95

Lepri, Daniela 334
Lescanne, Pierre 191

Madeira, Alexandre 340
Maeder, Christian 315
Malatesta, Lorenzo 19
Marsden, Daniel 205
Martins, Manuel A. 340
Maruyama, Yoshihiro 220
Mossakowski, Till 315
Mousavi, Mohammad Reza 36

Neves, Renato 340
Nipkow, Tobias 236
Nordvall Forsberg, Fredrik 19

Ölveczky, Peter Csaba 322, 334

Popescu, Andrei 236
Pous, Damien 34
Pretnar, Matija 1

Reniers, Michel A. 36
Rot, Jurriaan 95

Schröder, Lutz 253, 314
Silva, Alexandra 267
Sobociński, Paweł 282, 346
Stephens, Owen 346

Ţuţu, Ionuţ 299

Velebil, Jiří 51

Westerbaan, Bram 267
Winter, Joost 353

Zanasi, Fabio 80
Zhang, Min 328