Electronic Notes in Theoretical Computer Science

Mathematical Foundations of Programming Semantics

Twenty-ninth Annual Conference

Tulane University
New Orleans, LA USA
June 23 - 25, 2013

Guest Editors:

Dexter Kozen and Michael Mislove

Contents

Preface

This volume collects papers presented at the 29th Annual Conference on Mathematical Foundations of Programming Semantics (MFPS XXIX), held on the campus of Tulane University, New Orleans, USA, from Sunday, June 23 through Tuesday, June 25, 2013. MFPS was co-located with the 2013 Conference on Logic in Computer Science as well as the 2013 Computer Security Foundations Symposium.

The MFPS conferences are devoted to those areas of mathematics, logic, and computer science that are related to models of computation, in general, and to the semantics of programming languages, in particular. The series particularly stresses providing a forum where researchers in mathematics and computer science can meet and exchange ideas about problems of common interest. As the series also strives to maintain breadth in its scope, the conference strongly encourages participation by researchers in neighboring areas.

The Organizing Committee for MFPS consists of Andrej Bauer (Ljubljana), Stephen Brookes (CMU), Achim Jung (Birmingham), Catherine Meadows (NRL), Michael Mislove (Tulane), Joël Ouaknine (Oxford) and Prakash Panangaden (McGill). The local arrangements for MFPS XXIX were overseen by Michael Mislove.

The MFPS XXIX Program Committee members are:

Dexter Kozen (Cornell), *Chair*

Andrej Bauer (Ljubljana)

Nick Bezhanishvili (Utrecht)

Lars Birkedal (Aarhus)

Marcello Bonsangue (Leiden)

Stephen Brookes (CMU)

Venanzio Capretta (Nottingham)

Luca Cardelli, (Microsoft Research)

Volker Diekert (Stuttgart)

Dan Ghica (Birmingham)

Daniel Leivant (Indiana)

Catherine Meadows (NRL)

Paul-André Melliès (CNRS & Paris 7)

Michael Mislove (Tulane)

Carroll Morgan (New South Wales)

Paulo Oliva (Queen Mary London)

Luke Ong (Oxford)

Joël Ouaknine (Oxford)

Prakash Panangaden (McGill)

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: `www.elsevier.com/locate/entcs`

Jane Hillston (Edinburgh)

Radha Jagadeesan (DePaul)

Patricia Johann (Strathclyde)

Achim Jung (Birmingham)

Andrea Schalk (Manchester)

Phil Scott (Ottawa)

Ana Sokola (Salzburg)

James Worrell (Oxford)

The Program Committee selected 20 papers for presentation at the meeting. Regrettably a number of papers could not be accepted for lack of space in the conference program. The following people gave invited plenary talks at the meeting:

David Basin (ETH)

Jan Rutten (CWI)

Dana Scott (CMU)

In addition, there were two special sessions:

(i) A Special Session on Coalgebras, organized by Jan Rutten (CWI) and Alexandra Silva (Radboud). The session led off with Jan Rutten's plenary lecture. and included talks by Bart Jacobs (Radboud), Larry Moss (Indiana) and Lutz Schröder (Erlangen-Nürnburg). This session also comprised a LICS Tutorial Session.

(ii) A Special Session honoring Dana Scott on his 80th birthday, on *The Future of Programming Language Theory.* The session included talks by Andy Pitts (Cambridge), Steve Awodey (CMU), Robert Harper (CMU) and Andrej Bauer (Ljubljana), and concluded with Dana's plenary address. This was a joint MFPS-LICS session.

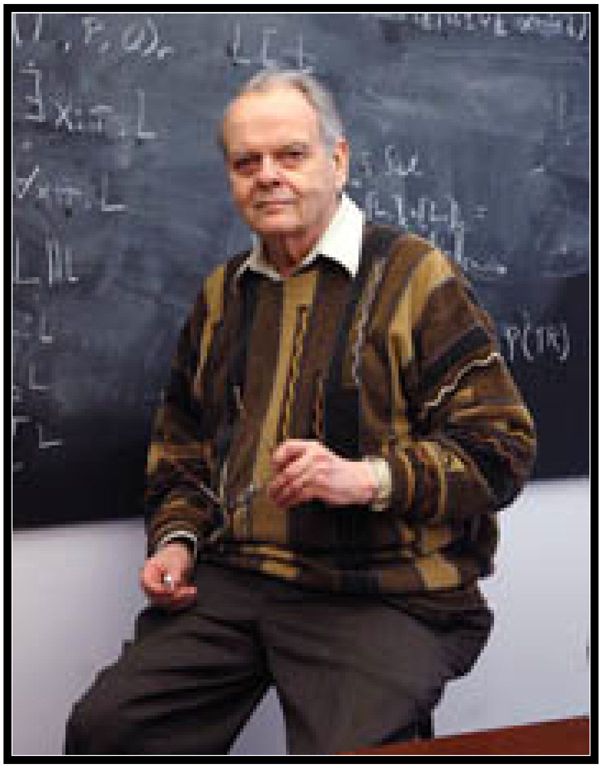
The program also included a Memorial Session for John Reynolds, to whom this volume is dedicated. The session included Uday Reddy's paper that was submitted to the conference, as well as short talks by Stephen Brookes (CMU), Benjamin Pierce (UPenn) and Gordon Plotkin (Edinburgh), and was chaired by Dana Scott (CMU).

It remains for us to thank all authors and speakers, the organizers of special sessions, the program committee, and the external referees for their contribution to the success of the conference.

We also would like to thank the US Office of Naval Research for its continued support of the MFPS series. Finally, we would like to thank the staff who helped with the preparations and execution of the conferences – Geralyn Caradona, Pam Philastre and Cammy Watts, Mathematics, and Debbie Ramil, Computer Science.

Dexter Kozen
Michael Mislove

Dedication



John C. Reynolds, June 1, 1935 – April 28, 2013

We dedicate these *Proceedings* of the Twenty-ninth Conference on the Mathematical Foundations of Programming Semantics to the memory of John C. Reynolds. John was strong supporter of this series, and of the community it represents. His colleagues will miss his careful scholarship, his generous mentoring and his inspiring leadership.

The Organizers

Abstracts of Invited Talks

Joint MFPS–LICS Session on Coalgebras

Automata and the algebra-coalgebra duality: on varieties and covarieties, on transition monoids and their dual, by Jan Rutten (CWI)

ABSTRACT: Because of the isomorphism

$$(X \times A) \longrightarrow X \;\simeq\; X \longrightarrow (A \longrightarrow X)$$

the transition structure of a deterministic automaton with state set X and with inputs from an alphabet A can be viewed both as an algebra [7] and as a coalgebra [14,15].

This algebra-coalgebra duality goes back to Arbib and Manes [2], who formulated it as a duality between reachability and observability, and is ultimately based on Kalman's [9] duality in systems theory between controllability and observability. Recently [4,5], it was used to give a new proof of Brzozowski's minimization algorithm for deterministic automata.

In this talk, we will discuss deterministic automata as an elementary and typical example for the combined use of algebraic and coalgebraic methods in computer science.

The algebra-coalgebra duality of automata will, more specifically, serve as a common perspective for the study [16] of both varieties and covarieties, which are classes of automata and languages defined by equations and coequations, respectively. Along the way, we will establish a first connection with Eilenberg's definition [7] of varieties of languages, which is based on the classical, algebraic notion of varieties of (transition) monoids.

This is joint work with Adolfo Ballester-Bolinches and Enric Cosme-Llópez (University of Valencia).

In addition to Jan Rutten's plenary talk, the session includes the following talks:

Coalgebra and Quantum Computing, by Bart Jacobs (Radboud)

ABSTRACT: Since a few years connections are established between the areas of coalgebra and quantum computing. This tutorial style talk will explore these connections, with focus on the probabilistic structures that are used in both areas: Kleisli categories of distribution/Giry monads and (commutative) C*-algebras.

Fractal Sets as Final Coalgebras Obtained by Completing an Initial Algebra, by Larry Moss (Indiana)

ABSTRACT: We are concerned with presentations of fractal sets in terms of final coalgebras. The first result on this topic was a theorem due to Peter Freyd: the unit interval $[0, 1]$ is the final coalgebra of a certain functor on the category of bipointed sets. Leinster (2011) offers a sweeping generalization of this result. He

is able to represent many self-similar spaces using (a) categorical bimodules, (b) non-degeneracy conditions on functors of various sorts; (c) a construction of final coalgebras for the types of functors of interest using a notion of resolution. His seminal paper also characterizes fractal sets as topological spaces.

Our major contribution is to suggest that in many cases of interest, point (c) above on resolutions is not needed in the construction of final coalgebras. Instead, one may obtain a number of spaces of interest as the Cauchy completion of an initial algebra, and this initial algebra is the set of points in a colimit of an omega-sequence of finite metric spaces. This generalizes the characterization in Hutchinson (1981) of fractal attractors as closures of the orbits of the critical points. In addition to simplifying the overall machinery, our work presents a metric space which is ”computationally related” to the overall fractal. For example, when applied to Freyd's construction, the initial algebra is the metric space of dyadic rational numbers in $[0, 1]$. Moreover, we have some results on the metric space aspects of this subject, a point raised in the 2010 MFPS paper of Hasuo, Jacobs, and Niqui.

This is joint work with Jayampathy Ratnayake and Robert Rose.

Coalgebraic Announcements, by Lutz Schröder (Erlangen–Nürnberg)

Abstract: The basic version of epistemic logic [8] has operators K_i read ‘agent i knows that’. As such, it is intended to deal with static snapshots of knowledge — it allows reasoning over a fixed state of knowledge of the individual agents but not over the evolution of that knowledge. Reasoning about the latter is the concern of *dynamic* epistemic logic, which includes operators that affect the knowledge of agents, typically increasing it. Maybe the most basic example of this type are *public announcements* [17] that make certain facts known to all agents involved, and in fact make them common knowledge in the sense that the agents know about each others knowledge of the announced fact, and know about everyone knowing that everybody knows etc.

Formally, public announcement logic (PAL) [13] is interpreted over Kripke models. The announcement formula $\langle\phi\rangle\psi$ states that the announced formula ϕ is true in the current state, and that in the restricted model in which only states satisfying ϕ are retained, ψ is true in the current state (which is retained by assumption). On the surface, this interpretation looks like a global transformation of the model, and this is also one of the perspectives discussed in previous coalgebraic work on the subject [3]. However, this view is not in keeping with the philosophy of coalgebraic logic at large, which is to take a *local* approach where possible, and as far as we know does not support generic algorithms. A different but evidently equivalent approach to the semantics of $\langle\phi\rangle$ is to change the underlying model (everywhere but) *locally*, that is, to modify only the set of successors at each state, limiting it to those successors that satisfy ϕ, and leave the set of states as such intact. Since this operation makes all states not satisfying ϕ unreachable, it clearly validates the same formulas as the original semantics.

In [6] we have taken this observation as the starting point for lifting the notion of *announcement* or, more neutrally, (model) *update* to the level of generality defined by coalgebraic logic. Coalgebraic logic serves as a generic framework for logics

with modal operators that covers a range of semantic features beyond the standard relational world. Typical examples include logics for probabilistic and weighted systems, neighbourhood structures, preferential structures (which appear in the semantics of conditional logics [10]), and game-oriented structures (which define the semantics of coalition logic [12] and alternating-time temporal logic [1]). Recall that coalgebraic logic parametrizes the semantics over the choice of a set functor T, whose coalgebras define the models under consideration, and interpretations of the modal operators $\heartsuit$ as predicate liftings for T, i.e. natural transformations $[\![\heartsuit]\!] : Q \to Q \circ T^{op}$ where Q denotes contravariant powerset.

We treat updates coalgebraically as modifications of the coalgebra map accumulated in the course of formula evaluation. The simplest setting here is one where we update the coalgebra map directly by means of natural transformations of the type

$$T \to (Q \twoheadrightarrow T)$$

where $(Q \twoheadrightarrow T)X := (TX)^{QX}$. Here, the second argument QX in this curried type represents the knowledge being announced, inducing a map from TX to TX which we can postcompose with the coalgebra map. The most well-behaved ones among these *updates* are those that satisfy a natural adjointness condition; this condition enables a translation of the language with updates into the modal base language. In the presence of a master modality, this translation can be modified into a polynomial satisfiability-preserving translation in the spirit of [11] so that complexity bounds can be inherited from the base language. We have then considered more general operations that transform predicates rather than the coalgebra structure itself. These exhibit a more complex behaviour; in particular, while it can be shown that it is always possible to translate the update language into a modal base language, this may require extending the modal signature so that one needs to harness the arising closure process manually.

In the talk we will give an introduction to coalgebraic logic in general and then discuss example applications of the above principle. These include announcements in settings involving uncertainty and in weak epistemic logics equipped with a neighbourhood semantics in the style of [18], various type of lossy or fallible announcements, and games with changing rules.

This is joint work with Facundo Carreiro and Daniel Gorin.

Joint MFPS–LICS Session on the Future of Programming Language Theory

Stochastic Lambda-Calculi, by Dana S. Scott (CMU and UC Berkeley)

Abstract: Many authors have suggested ways of adding random elements and probability assessments to versions of Church's Lambda-Calculus. The speaker realized recently that the so-called Graph Model based on enumeration operators acting on the powerset of the integers could easily be expanded by the adjunction of random variables. Indeed, using any countably based continuous lattice L, the cartesian power $L^{\mathbb{N}}$ can be very quickly be made into a lambda-calculus model, and random variables can always be naturally adjoined. The talk will explore possible

applications to probabilistic reasoning, randomized algorithms, and fuzzy logic.

In addition to Dana Scott's plenary address, the session includes the following talks:

Symmetric Scott, by Andy Pitts (Cambridge)

Abstract: Dana Scott has shown us that an effective way to make progress in programming language theory is mathematization: formalize a programming concept sufficiently for it to become a piece of mathematics. Then sometimes an existing mathematical theory can usefully be applied and sometimes a new one has to be invented.

This talk is about the mathematization of a ubiquitous feature of programming languages – constructs involving binding and localising names. Nominal sets provide a mathematical theory of structures involving names, based on some existing, but subtle ideas to do with symmetry and support. What is emerging is a computation theory for structures that are infinite, but finite modulo symmetry. It involves "symmetry-aware" versions of several topics to which Dana has contributed: set theory, automata theory, domain theory. I will concentrate in particular on computation at higher types and a symmetry-aware version of Scott domains.

Constructing Higher Inductive Types in Homotopy Type Theory, by Steve Awodey (CMU)

Abstract: One of the most interesting new developments in HoTT is the use of "higher inductive types" to provide direct, logical descriptions of some important mathematical spaces and constructions: spheres, cell complexes, truncations, quotients. In type theory, one has the so-called "impredicative encodings" of conventional inductive types such the natural numbers, albeit generally with only weak recursion principles. Using the basic idea of intensional identity types as path spaces, we show how to strengthen these familiar recursion principles for conventional inductive types, and then how to extend them to some higher inductive types. In particular, we give an impredicative encoding of the unit circle S1. A realizability model based on modest groupoids is given, in order to establish the consistency of impredicative HoTT.

Unifying Programming Language Semantics with Algorithm Analysis, by Robert Harper (CMU)

Abstract: The theory of programming languages and the theory of algorithms are highly developed area of computer science with notable influence on the practical problems of software development. The theory of languages emphasizes the structural aspects of programming, principally the central problem of composition of programs from components. The theory of algorithms emphasizes the complexity aspects, principally the analysis of the asymptotic time efficiency of a program. Compared with the depth of their separate development, relatively little has been done to integrate these two theories of programming.

One reason for this separation is that algorithmic theory is based on low-level models of computation, such as the Turing machine or the RAM, that provide a clear notion of time complexity measured as the number of steps executed by the machine to complete a computation. But such low-level models provide no support for composition of programs from components, and no notion of abstraction with which to express high-level program structure. Conversely, programming language theory is based on high-level models, principally the λ-calculus, that directly address problems of compositionality, but neglect the analysis of complexity.

This talk is concerned with the integration of programming language semantics with algorithm analysis through the use of a cost semantics, which provides an abstract definition of programming languages at a level suitable for addressing compositionality while also admitting the asymptotic analysis of their complexity. A prime example of the effective use of cost semantics is the treatment of parallel computation, which is entirely a matter of efficiency that can be achieved only at a high-level of abstraction. Another example is the use of cost semantics to analyze the I/O complexity of a program, a measure of the cost of transferring data from secondary to primary storage, a very significant factor for real-world performance.

[This talk represents joint work with Guy Blelloch of Carnegie Mellon University.]

References

[1] Alur, R., T. A. Henzinger and O. Kupferman, *Alternating-time temporal logic*, J. ACM **49** (2002), pp. 672–713.

[2] M.A. Arbib and E.G. Manes, Adjoint machines, state-behaviour machines, and duality, Journal of Pure and Applied Algebra, 6:313–344, 1975.

[3] Baltag, A., *A coalgebraic semantics for epistemic programs*, in: *Coalgebraic Methods in Computer Science*, ENTCS **82** (2003), pp. 17–38.

[4] F. Bonchi, M. Bonsangue, J. Rutten, and A. Silva, Brzozowski's algorithm (co)algebraically. In Logic and Program Semantics, volume 7230 of LNCS, pages 12–23, 2012.

[5] F. Bonchi, M. Bonsangue, H. Hansen, P. Panangaden, J. Rutten, and A. Silva, Algebra-coalgebra duality in Brzozowski's minimization algorithm. Submitted.

[6] Carreiro, F., D. Gorín and L. Schröder, *Coalgebraic announcement logics*, in: *Proc. 40th International Colloquium on Automata, Languages and Programming, ICALP 2013*, LNCS **7966** (2013), p. 101112, to appear.

[7] S.Eilenberg. Automata, languages and machines (Volumes A and B). Pure and applied mathematics. Academic Press, 1974 and 1976.

[8] Hintikka, J., "Knowledge and belief," Cornell University Press, 1962.

[9] R. Kalman. On the general theory of control systems, IRE Transactions on Automatic Control, 4(3):110–110, 1959.

[10] Lewis, D., "Counterfactuals," Harvard University Press, 1973.

[11] Lutz, C., *Complexity and succinctness of public announcement logic*, in: *Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2006, pp. 137–143.

[12] Pauly, M., *A modal logic for coalitional power in games*, J. Logic Comput. **12** (2002), pp. 149–166.

[13] Plaza, J. A., *Logics of public communications*, in: *International Symposium on Methodologies for Intelligent Systems*, 1989, pp. 201–216.

[14] J.J.M.M. Rutten, Automata and coinduction (an exercise in coalgebra), Proceedings of CONCUR'98. Volume 1466 of LNCS, pages 194–218, 1998.

[15] J.J.M.M. Rutten, Universal coalgebra: a theory of systems, Theoretical Computer Science, 249(1):3–80, 2000.

[16] J.Rutten, A. Ballester-Bolinches, E. Cosme-Llópez, Varieties and covarieties of languages (extended abstract), These *Proceedings.*

[17] van Ditmarsch, H., W. van der Hoek and B. Kooi, "Dynamic epistemic logics," Springer, 2007.

[18] Vardi, M., *On the complexity of epistemic reasoning*, in: *Logic in Computer Science, LICS 89*, IEEE, 1989, pp. 243–251.

Varieties and Covarieties of Languages (Preliminary Version)

Jan Rutten[a,1] Adolfo Ballester-Bolinches[b,2]
Enric Cosme-Llópez[b,3]

[a] *CWI and Radboud University*
Amsterdam, The Netherlands

[b] *Departament d'Àlgebra*
Universitat de València
València, Spain

Abstract

Because of the isomorphism $(X \times A) \to X \cong X \to (A \to X)$, the transition structure of a deterministic automaton with state set X and with inputs from an alphabet A can be viewed both as an algebra and as a coalgebra. This algebra-coalgebra duality goes back to Arbib and Manes, who formulated it as a duality between reachability and observability, and is ultimately based on Kalman's duality in systems theory between controllability and observability. Recently, it was used to give a new proof of Brzozowski's minimization algorithm for deterministic automata. Here we will use the algebra-coalgebra duality of automata as a common perspective for the study of both varieties and covarieties, which are classes of automata and languages defined by equations and coequations, respectively. We make a first connection with Eilenberg's definition of varieties of languages, which is based on the classical, algebraic notion of varieties of (transition) monoids.

Keywords: Automata, variety, covariety, equation, coequation, algebra, coalgebra.

1 Introduction

Because of the isomorphism

$$(X \times A) \to X \quad \cong \quad X \to (A \to X)$$

the transition structure of a deterministic automaton with state set X and with inputs from an alphabet A can be viewed both as an algebra [11] and as a coalgebra [19,20]. As a consequence, both the algebraic notion of *variety* and the coalgebraic notion of *covariety* apply. In this paper, we present a preliminary version of what is

[1] Email: janr@cwi.nl

[2] Email: Adolfo.Ballester@uv.es

[3] Email: enric.cosme@uv.es

to become a systematic study of varieties and covarieties of automata and of formal languages.

We will define a variety of automata (viewed as algebras) in the usual way, as a class defined by *equations* [12]. A covariety of automata (viewed as coalgebras) will be a class defined by *coequations* [20]. Varieties and covarieties of automata will then be used to define varieties and covarieties of *languages*. Our notion of a variety of languages is different from the classical definition by Eilenberg [12,18], and we will make some initial observations on how the two notions are related.

The setting of our investigations will be the following picture:

$$\begin{tikzcd}
1 \arrow[d] \arrow[dr, bend left, "x"] & & 2 \\
A^* \arrow[r, dashed, "r_x"'] & X \arrow[ur, bend left, "c"] \arrow[r, dashed, "o_c"'] & 2^{A^*} \arrow[u]
\end{tikzcd} \tag{1}$$

(This diagram will be explained in more detail in Section 3.) In the middle, we have the state set X of a given automaton. On the left, A^* is the set of all words over A, and on the right, 2^{A^*} is the set of all languages over A. For every choice of a *point* (initial state) $x \in X$, the function r_x sends any word w to the state x_w reached from x on input w. And for every choice of a *colouring* (set of final states) $c : X \to 2$, the function o_c sends any state to the language it accepts.

Both the pointed automata A^* (with the empty word as point) and X with point x, are algebras. And both the coloured automata 2^{A^*} (with colouring as explained later) and X with colouring c, are coalgebras. The unique existence of the function (in fact, a homomorphism of algebras) r_x is induced by the *initiality* of A^*. And the unique existence of the function (a homomorphism of coalgebras) o_c is induced by the *finality* of 2^{A^*}.

(Sets of) equations will live in the left – algebraic – part of our diagram; in short, they correspond to *quotients* of A^*. And (sets of) coequations live in the right – coalgebraic – part of our diagram; they will correspond to *subautomata* of 2^{A^*}. As a consequence, diagram (1) allows us to define both varieties and covarieties, and to study their properties from a common perspective.

The *algebra-coalgebra duality* of diagram (1) is a modern rendering of the duality between *reachability* and *observability* of automata [2,1], which ultimately goes back to Kalman's duality between controllability and observability in system theory [14,15]. (See also [7,9] for further categorical generalisations.)

Recently [6,3], this algebra-coalgebra duality of automata was used to give a new proof and various generalisations of Brzozowski's minimization algorithm [8]. The present work goes in a different direction, focusing on (co)equations and (co)varieties. Notably, we will further refine diagram (1) as follows:

$$\begin{tikzcd}
1 \arrow[d] \arrow[dr, bend left] \arrow[drr, bend left, "x"] & & & & 2 \\
A^* \arrow[r, "r"] \arrow[rr, bend right, "r_x"'] & \mathsf{free}(X,\alpha) \arrow[r] & X \arrow[r] \arrow[rr, bend right, "o_c"'] \arrow[urr, bend left, "c"] & \mathsf{cofree}(X,\alpha) \arrow[r, "o"] \arrow[ur, bend left] & 2^{A^*} \arrow[u]
\end{tikzcd}$$

(For details, see Section 5.) The new diagram includes, for every automaton X with transition function $\alpha : X \to X^A$, the (pointed) automaton $\mathsf{free}(X, \alpha)$, which represents the *largest set of equations* satisfied by (X, α). And, dually, we will construct a (coloured) automaton $\mathsf{cofree}(X, \alpha)$, which represents the *smallest set of coequations* satisfied by (X, α).

We already mentioned above that our definition of a variety of languages is different from Eilenberg's, which is derived from the (classical, algebraic) notion of variety of *monoids*. A first step towards an understanding of the relation between the classical and the present notion of variety consists of the – elementary but to us somewhat surprising – observation that $\mathsf{free}(X, \alpha)$ is isomorphic to the so-called *transition monoid* of X (which is called the *syntactic* monoid in case X is minimal) [18]. This observation furthermore implies that the coloured automaton $\mathsf{cofree}(X, \alpha)$ can be viewed as a dual version of the transition monoid.

Much remains to be further understood. We already mentioned the connection with Eilenberg's variety theorem. Furthermore, we would also like to relate the present algebra-coalgebra perspective to recent developments on varieties of languages, notably [13] and [4,5]. Finally, it should be possible to generalise the present setting, along the lines of [6,3], from deterministic automata to other structures such as Mealy machines, weighted automata etc.

2 Preliminaries

Let A be a finite alphabet, in all our examples fixed to $\{a, b\}$. We write A^* for the set of all finite sequences (words) over A. We denote the empty word by ε and the concatenation of two words v and w by vw.

For sets X and Z we define $X^Z = \{g \mid g : Z \to X\}$. For sets X, Y, Z and functions $f : X \to Y$ we define $f^Z : X^Z \to Y^Z$ by $f^Z(g) = f \circ g$.

We define the *image* and the *kernel* of a function $f : X \to Y$ by

$$\mathsf{im}(f) = \{y \in Y \mid \exists x \in X,\ f(x) = y\}$$

$$\mathsf{ker}(f) = \{(x_1, x_2) \in X \times X \mid f(x_1) = f(x_2)\}$$

A *language* L over A is a subset $L \subseteq A^*$ and we denote the set of all languages over A by

$$2^{A^*} = \{L \mid L \subseteq A^*\}$$

(ignoring here and sometimes below the difference between subsets and characteristic functions). For a language $L \subseteq A^*$ and $a \in A$ we define the *a-derivative* of L by

$$L_a = \{v \in A^* \mid av \in L\}$$

and we define, more generally,

$$L_w = \{v \in A^* \mid wv \in L\}$$

We define the *initial value* $L(0)$ of L by

$$L(0) = \begin{cases} 1 \text{ if } \varepsilon \in L \\ 0 \text{ if } \varepsilon \notin L \end{cases}$$

For a functor $F : Set \to Set$, an *F-algebra* is a pair (S, α) consisting of a set S and a function $\alpha : F(S) \to S$. An *F-coalgebra* is a pair (S, α) with $\alpha : S \to F(S)$.

We will be using the following functors:

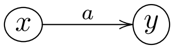
$$\begin{aligned} F(S) &= S^A \\ G(S) &= S \times A \\ (2 \times F)(S) &= 2 \times S^A \\ (1 + G)(S) &= 1 + (S \times A) \end{aligned}$$

Automata

An *automaton* is a pair (X, α) consisting of a (possibly infinite) set X of states and a transition function

$$\alpha : X \to X^A$$

In pictures, we use the following notation:



$$\Leftrightarrow \qquad \alpha(x)(a) = y$$

We will also write $x_a = \alpha(x)(a)$ and, more generally,

$$x_\varepsilon = x \qquad x_{wa} = \alpha(x_w)(a)$$

We observe that automata are *F-coalgebras*. Because there is, for any A and X, an isomorphism

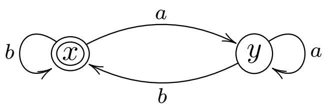
$$(\tilde{\ }) : \ (X \to X^A) \to ((X \times A) \to X) \qquad \tilde{\alpha}(x, a) = \alpha(x)(a)$$

automata are also *G-algebras* [17].

An automaton can be decorated by means of a *colouring* function

$$c : X \to 2$$

using a basic set of colours $2 = \{0, 1\}$. We call a state x *accepting* (or final) if $c(x) = 1$, and non-accepting if $c(x) = 0$. We call a triple (X, c, α) a *coloured* automaton. In pictures, we use a double circle to indicate that a state is accepting. For instance, in the following automaton



the state x is accepting and the state y is not.

By pairing the functions c and α, we see that coloured automata are $(2 \times F)$-coalgebras:

$$\langle c, \alpha \rangle : X \to 2 \times X^A$$

An automaton can also have an *initial state* $x \in X$, here represented by a function

$$x : 1 \to X$$

where $1 = \{0\}$. We call a triple (X, x, α) a *pointed* automaton. By pairing the functions x and $\tilde{\alpha}$, we see that pointed automata are $(1 + G)$-algebras:

$$[x, \tilde{\alpha}] : (1 + (X \times A)) \to X$$

We call a 4-tuple (X, x, c, α) a *pointed and coloured automaton.* We could depict it by either of the two following diagrams

$$\begin{tikzcd}
1 \arrow[dr, bend left, "x"] & & 2 & & 1 \arrow[dr, bend left, "x"] & & 2 \\
& X \arrow[ur, bend left, "c"] \arrow[d, "\alpha"'] & & & & X \arrow[ur, bend left, "c"] & \\
& X^A & & & & X \times A \arrow[u, "\tilde{\alpha}"] &
\end{tikzcd}$$

We will be using the diagram on the left, which is just a matter of personal preference.

We observe further that pointed and coloured automata are simply called *automata* in most of the literature on automata theory. A pointed and coloured automaton (X, x, c, α) is neither an algebra nor a coalgebra – because of c and x, respectively – which can be a cause of fascination and confusion alike.

Homomorphisms, subautomata, bisimulations

A function $h : X \to Y$ is a *homomorphism* between automata (X, α) and (Y, β) if it makes the following diagram commute:

$$\begin{tikzcd}
X \arrow[r, "h"'] \arrow[d, "\alpha"'] & Y \arrow[d, "\beta"] \\
X^A \arrow[r, "h^A"'] & Y^A
\end{tikzcd}$$

A homomorphism of pointed automata (X, x, α) and (Y, y, β) and of coloured automata (X, c, α) and (Y, d, β) moreover respects initial values and colours, respectively:

$$\begin{tikzcd}
1 \arrow[d, "x"'] \arrow[dr, bend left, "y"] & & & & 2 \\
X \arrow[r, "h"'] & Y & & X \arrow[ur, bend left, "c"] \arrow[r, "h"'] & Y \arrow[u, "d"']
\end{tikzcd}$$

If in the diagrams above $X \subseteq Y$, and (i) h is subset inclusion

$$h : X \subseteq Y$$

(and, moreover (ii) $x = y$ or (iii) $c = d$), then we call X a (i) *subautomaton* of Y (respectively (ii) *pointed* and (iii) *coloured subautomaton*).

For an automaton (X, α) and $x \in X$, the *subautomaton generated by* x, denoted by

$$\langle x \rangle \subseteq X$$

consists of the smallest subset of X that contains x and is closed under transitions.

We call a relation $R \subseteq X \times Y$ a *bisimulation of automata* if for all $(x, y) \in X \times Y$,

$$(x, y) \in R \;\;\Rightarrow\;\; \forall a \in A,\; (x_a, y_a) \in R$$

(where $x_a = \sigma(x)(a)$ and $y_a = \tau(y)(a)$). For pointed automata (X, x, α) and (Y, y, β), R is a *pointed bisimulation* if, moreover, $(x, y) \in R$. And for coloured automata (X, x, α) and (Y, y, β), R is a *coloured bisimulation* if, moreover, for all $(x, y) \in X \times Y$,

$$(x, y) \in R \;\;\Rightarrow\;\; c(x) = d(y)$$

A bisimulation $E \subseteq X \times X$ is called a bisimulation *on* X. If E is an equivalence relation then we call it a *bisimulation equivalence.* The quotient map of a bisimulation equivalence on X is a homomorphism of automata:

$$\begin{tikzcd}
X \arrow[r, "q"] \arrow[d, "\alpha"'] & X/E \arrow[d, "{[\alpha]}"] \\
X^A \arrow[r, "q^A"'] & (X/E)^A
\end{tikzcd}$$

with the obvious definitions of X/E, q and $[\alpha]$. If the equivalence E is a pointed bisimulation on (X, x, α) or a coloured bisimulation on (X, c, α), then we moreover have, respectively,

$$\begin{tikzcd}
1 \arrow[d, "x"'] \arrow[dr, bend left, "{[x]}"] & \\
X \arrow[r, "h"'] & X/E
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
& 2 \\
X \arrow[ur, bend left, "c"] \arrow[r, "h"'] & X/E \arrow[u, "{[c]}"']
\end{tikzcd}$$

with, again, the obvious definitions of $[x]$ and $[c]$.

For a homomorphism $h : X \to Y$, $\mathsf{ker}(h)$ is a bisimulation equivalence on X and $\mathsf{im}(h)$ is a subautomaton of Y. Any homomorphism h factors through quotient and inclusion homomorphisms as follows:

$$\begin{tikzcd}
X \arrow[r, "q"'] \arrow[rr, bend left, "h"] \arrow[d, "\alpha"'] & X/\mathsf{ker}(h) \arrow[r, "i"'] \arrow[d, "{[\alpha]}"] & Y \arrow[d, "\beta"] \\
X^A \arrow[r, "q^A"] \arrow[rr, bend right, "h^A"'] & (X/\mathsf{ker}(h))^A \arrow[r, "i^A"] & Y^A
\end{tikzcd}$$

Note that $X/\mathsf{ker}(h) \cong \mathsf{im}(h)$. Because q is surjective and i is injective, the pair (q, i) is called an *epi-mono factorisation* of h.

Congruence relations

A *right congruence* is an equivalence relation $E \subseteq A^* \times A^*$ such that, for all $(v, w) \in A^* \times A^*$,

$$(v, w) \in E \;\;\Rightarrow\;\; \forall u \in A^*,\; (vu,\, wu) \in E$$

A *left congruence* is an equivalence relation $E \subseteq A^* \times A^*$ such that, for all $(v, w) \in A^* \times A^*$,

$$(v, w) \in E \;\;\Rightarrow\;\; \forall u \in A^*,\; (uv,\, uw) \in E$$

We call E a *congruence* if it is both a right and a left congruence. Note that E is a right congruence iff it is a bisimulation equivalence on (A^*, σ).

Products and coproducts of automata

Automata (are both G-algebras and F-coalgebras and hence) have both products and coproducts, as follows.

- The *product* of two automata (X, α) and (Y, β) is given by $(X \times Y, \gamma)$ where $X \times Y$ is the Cartesian product and where

$$\gamma : (X \times Y) \to (X \times Y)^A \qquad \gamma((x, y))(a) = (\,\alpha(x)(a),\, \beta(y)(a)\,)$$

- The *coproduct* (or: sum) of two automata (X, α) and (Y, β) is given by $(X + Y, \gamma)$ where $X + Y$ is the disjoint union and where

$$\gamma : (X + Y) \to (X + Y)^A \qquad \gamma(z)(a) = \begin{cases} \alpha(z)(a) \text{ if } z \in X \\ \beta(z)(a) \text{ if } z \in Y \end{cases}$$

Pointed automata (are $(1 + G)$-algebras and hence) have products, as follows. The product of two pointed automata (X, x, α) and (Y, y, β) is given by $(X \times Y, (x, y), \gamma)$ with $(X \times Y, \gamma)$ as above and with initial state

$$(x, y) : 1 \to X \times Y$$

Coloured automata (are $(2 \times F)$-coalgebras and hence) have coproducts, as follows. The coproduct of two coloured automata (X, c, α) and (Y, d, β) is given by $(X + Y, [c, d], \gamma)$ with $(X + Y, \gamma)$ as above and with colouring function

$$[c, d] : (X + Y) \to 2 \qquad [c, d](z) = \begin{cases} c(z) \text{ if } z \in X \\ d(z) \text{ if } z \in Y \end{cases}$$

All of the above binary (co)products can be easily generalised to (co)products of arbitrary families of automata.

3 Setting the scene

The set A^* forms a pointed automaton $(A^*, \varepsilon, \sigma)$ with initial state ε and transition function σ defined by

$$\sigma : A^* \to (A^*)^A \qquad \sigma(w)(a) = wa$$

It is *initial* in the following sense: for any given automaton (X, α), every choice of initial state $x : 1 \to X$ induces a unique function $r_x : A^* \to X$, given by $r_x(w) = x_w$, that makes the following diagram commute:

$$\begin{tikzcd}
1 \arrow[d, "\varepsilon"'] \arrow[dr, bend left, "x"] & \\
A^* \arrow[r, dashed, "r_x"'] \arrow[d, "\sigma"'] & X \arrow[d, "\alpha"] \\
(A^*)^A \arrow[r, dashed, "(r_x)^A"'] & X^A
\end{tikzcd}$$

This property makes $(A^*, \varepsilon, \sigma)$ an *initial* $(1+G)$*-algebra*. Equivalently, the automaton (A^*, σ) is a G*-algebra that is free on the set* 1. The function r_x maps a word w to the state x_w reached from the initial state x on input w and is therefore called the *reachability* map for (X, x, α).

The set 2^{A^*} of languages forms a coloured automaton $(2^{A^*}, \varepsilon?, \tau)$ with colouring function $\varepsilon?$ defined by

$$\varepsilon? \; : \; 2^{A^*} \to 2 \qquad \varepsilon?(L) = L(0)$$

and transition function τ defined by

$$\tau : 2^{A^*} \to (2^{A^*})^A \qquad \tau(L)(a) = L_a$$

It is *final* in the following sense: for any given automaton (X, α), every choice of colouring function $c : X \to 2$ induces a unique function $o_c : X \to 2^{A^*}$, given by $o_c(x) = \{w \mid c(x_w) = 1\}$, that makes the following diagram commute:

$$\begin{tikzcd}
 & 2 \\
X \arrow[ur, bend left, "c"] \arrow[r, dashed, "o_c"'] \arrow[d, "\alpha"'] & 2^{A^*} \arrow[u, "\varepsilon?"'] \arrow[d, "\tau"] \\
X^A \arrow[r, dashed, "(o_c)^A"'] & (2^{A^*})^A
\end{tikzcd}$$

This property makes $(2^{A^*}, \varepsilon?, \tau)$ a *final* $(2 \times F)$*-coalgebra*. Equivalently, the automaton $(2^{A^*}, \tau)$ is an F*-coalgebra that is cofree on the set* 2. The function o_c maps a state x to the language $o_c(x)$ accepted by x. Since the language $o_c(x)$ can be viewed as the observable behaviour of x, the function o_c is called the *observability* map.

The scene

Summarizing, we have set the following scene for our investigations:

$$\begin{tikzcd}
1 \arrow[d, "\varepsilon"'] \arrow[dr, bend left, "x"] & & 2 \\
A^* \arrow[r, dashed, "r_x"'] \arrow[d, "\sigma"'] & X \arrow[r, dashed, "o_c"'] \arrow[d, "\alpha"'] \arrow[ur, bend left, "c"] & 2^{A^*} \arrow[u, "\varepsilon?"'] \arrow[d, "\tau"] \\
(A^*)^A \arrow[r, dashed, "(r_x)^A"'] & X^A \arrow[r, dashed, "(o_c)^A"'] & (2^{A^*})^A
\end{tikzcd} \tag{2}$$

If the reachability map r_x is *surjective* then we call (X, x, α) *reachable.* If the observability map o_c is *injective* then we call (X, c, α) *observable.* And if r_x is surjective *and* o_c is injective then we call (X, x, c, α) (reachable and observable, or:) *minimal.*

For a given language $L \in 2^{A^*}$, there is the following variation of the picture above:

$$\begin{tikzcd}
1 \arrow[d, "\varepsilon"'] \arrow[dr, bend left, "L"] & \\
A^* \arrow[r, "h"] \arrow[dr, bend right, "L"'] & 2^{A^*} \arrow[d, "\varepsilon?"] \\
 & 2
\end{tikzcd}$$

where the lower L is in fact the characteristic function of $L \subseteq A^*$, and where the homomorphism h satisfies $h = r_L = o_L$ and $h(w) = L_w$. As a consequence, we have $h(v) = h(w)$ iff

$$\forall u \in A^*,\ vu \in L \Leftrightarrow wu \in L$$

which we recognise as the celebrated *Myhill-Nerode* equivalence. A *minimal automaton accepting* L is now obtained by the epi-mono factorisation of h:

$$\begin{tikzcd}
1 \arrow[d, "\varepsilon"'] \arrow[dr, bend left, "x"] \arrow[drr, bend left, "L"] & & \\
A^* \arrow[r, "q"] \arrow[drr, bend right, "L"'] & A^*/\mathsf{ker}(h) \arrow[r, "i"] \arrow[dr, bend right, "c"] & 2^{A^*} \arrow[d, "\varepsilon?"] \\
 & & 2
\end{tikzcd}$$

where $x = q \circ \varepsilon$ and $c = \varepsilon? \circ i$. This minimal automaton is unique up-to isomorphism because epi-mono factorisations are. And because $A^*/\mathsf{ker}(h) \cong \mathsf{im}(h)$, it is equal to

$$\langle L \rangle \subseteq 2^{A^*}$$

that is, the subautomaton of $(2^{A^*}, \varepsilon?)$ generated by L.

In conclusion of this section, we observe that $\langle L \rangle$ is finite iff the language L is *rational.* This fact is a version [8,10] of Kleene's correspondence between finite automata and rational languages [16].

4 Equations and coequations

We will be referring to the situation of (2).

Definition 4.1 [equations] A *set of equations* is a bisimulation equivalence relation $E \subseteq A^* \times A^*$ on the automaton (A^*, σ). We define $(X, x, \alpha) \models E$ – and say: *the pointed automaton* (X, x, α) *satisfies* E – by

$$(X, x, \alpha) \models E \quad \Leftrightarrow \quad \forall (v, w) \in E,\ x_v = x_w$$

Because

$$\forall (v, w) \in E,\ x_v = x_w \quad \Leftrightarrow \quad E \subseteq \mathsf{ker}(r_x)$$

we have, equivalently, that $(X, x, \alpha) \models E$ iff the reachability map r_x factors through A^*/E:

$$\begin{tikzcd}
1 \arrow[d, "\varepsilon"'] \arrow[dr, bend left, "{[\varepsilon]}"'] \arrow[drr, bend left, "x"] & & \\
A^* \arrow[r, "q"] \arrow[rr, bend right, "r_x"'] & A^*/E \arrow[r, "h"] & X
\end{tikzcd}$$

where the homomorphisms (of pointed automata) q and h are given by

$$q(w) = [w] \qquad\qquad h([w]) = r_x(w)$$

We define $(X, \alpha) \models E$ – and say: *the automaton* (X, α) *satisfies* E – by

$$\begin{aligned}(X, \alpha) \models E &\Leftrightarrow \forall x : 1 \to X,\ \ (X, x, \alpha) \models E \\ &\Leftrightarrow \forall x \in X,\ \forall (v, w) \in E,\ x_v = x_w\end{aligned}$$

□

Note that we consider *sets* of equations E and that $(v, w) \in E$ implies $(vu, wu) \in E$, for all $v, w, u \in A^*$, because E is – by definition – a bisimulation relation on (A^*, σ). Still we shall sometimes consider also *single* equations $(v, w) \in A^* \times A^*$ and use the following shorthand:

$$(X, x, \alpha) \models\ v = w \quad \Leftrightarrow \quad (X, x, \alpha) \models E_{v=w}$$

where $E_{v=w}$ is defined as the smallest bisimulation equivalence on A^* containing (v, w). We shall use also variations such as

$$(X, x, \alpha) \models\ \{v = w,\, t = u\} \quad \Leftrightarrow \quad (X, x, \alpha) \models\ v = w \ \ \wedge\ \ (X, x, \alpha) \models\ t = u$$

Definition 4.2 [coequations]

A *set of coequations* is a subautomaton $D \subseteq 2^{A^*}$ of the automaton $(2^{A^*}, \tau)$. We define $(X, c, \alpha) \models D$ – and say: *the coloured automaton* (X, c, α) *satisfies* D – by

$$(X, c, \alpha) \models D \quad \Leftrightarrow \quad \forall x \in X,\ o_c(x) \in D$$

Because

$$\forall x \in X,\ o_c(x) \in D \quad \Leftrightarrow \quad \mathsf{im}(o_c) \subseteq D$$

we have, equivalently, that $(X, c, \alpha) \models D$ iff the observability map o_c factors through D:

$$\begin{tikzcd}
 & & 2 \\
X \arrow[rru, bend left, "c"] \arrow[r, "h"] \arrow[rr, bend right, "o_c"'] & D \arrow[ru, bend left, "\varepsilon?"] \arrow[r, "i"] & 2^{A^*} \arrow[u, "\varepsilon?"']
\end{tikzcd}$$

where the homomorphisms (of coloured automata) h and i are given by

$$h(x) = o_c(x) \qquad\qquad i(L) = L$$

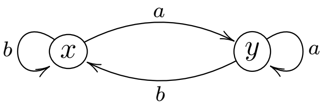
We define $(X, \alpha) \models D$ – and say: *the automaton* (X, α) *satisfies* D – by

$$\begin{aligned}(X, \alpha) \models D &\Leftrightarrow \forall c : X \to 2,\ \ (X, c, \alpha) \models D \\ &\Leftrightarrow \forall c : X \to 2,\ \forall x \in X,\ o_c(x) \in D\end{aligned}$$

□

Example 4.3 We consider the automaton (Z, γ) defined by the following diagram:

$(Z, \gamma) \quad =$



Here are some examples of equations:

$$\begin{aligned}(Z, x, \gamma) &\models \{b = \varepsilon,\ ab = \varepsilon,\ aa = a\} \\ (Z, y, \gamma) &\models \{a = \varepsilon,\ ba = \varepsilon,\ bb = b\}\end{aligned}$$

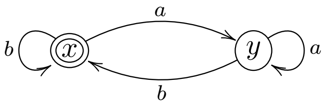
Taking the intersection of the (bisimulation equivalences generated by) these sets, we obtain that

$$(Z, \gamma) \models \ \{aa = a,\ bb = b,\ ab = b,\ ba = a\}$$

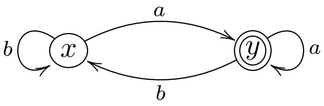
The above set of equations or, again more precisely, the bisimulation equivalence relation on (A^*, σ) generated by it, is the largest set of equations satisfied by (Z, γ).

For examples of coequations, we consider the following 2 (out of all 4 possible) coloured versions of (Z, γ):

$(Z, c, \gamma) =$

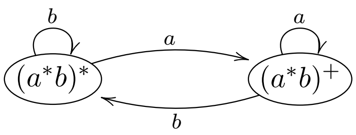


$(Z, d, \gamma) =$

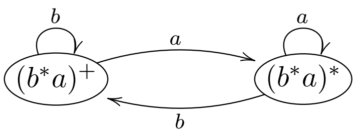


(Thus $c(x) = 1$, $c(y) = 0$, $d(x) = 0$ and $d(y) = 1$.) The observability mappings o_c and o_d map these automata to

$\mathsf{im}(o_c) =$



$\mathsf{im}(o_d) =$



It follows that

$$(Z, c, \gamma) \models \ \{(a^*b)^*,\ (a^*b)^+\} \qquad\qquad (Z, d, \gamma) \models \ \{(b^*a)^*,\ (b^*a)^+\}$$

□

5 Free and cofree automata

Let (X, α) be an arbitrary automaton. We show how to construct an automaton that corresponds to the *largest set of equations* satisfied by (X, α). And, dually, we construct an automaton that corresponds to the *smallest set of coequations* satisfied by (X, α). For notational convenience, we assume X to be finite but nothing will depend on that assumption.

Definition 5.1 [free automaton, $\mathsf{Eq}(X, \alpha)$] Let $X = \{x_1, \ldots, x_n\}$ be the set of states of a finite automaton (X, α). We define a pointed automaton $\mathsf{free}(X, \alpha)$ in two steps, as follows:

(i) First we take the product of the n pointed automata (X, x_i, α) that we obtain by letting the initial element x_i range over X. This yields a pointed automaton $(\Pi X, \bar{x}, \bar{\alpha})$ with

$$\Pi X = \prod_{x:1 \to X} X_x \quad \cong \quad X^n$$

(where $X_x = X$), with $\bar{x} = (x_1, \ldots, x_n)$, and with $\bar{\alpha} : \Pi X \to (\Pi X)^A$ defined by

$$\bar{\alpha}(y_1, \ldots, y_n)(a) = ((y_1)_a, \ldots, (y_n)_a)$$

(ii) Next we define $(\mathsf{free}(X, \alpha), \bar{x}, \bar{\alpha})$ by $\mathsf{free}(X, \alpha) = \mathsf{im}(r_{\bar{x}})$, where $r_{\bar{x}}$ is the reachability map for $(\Pi X, \bar{x}, \bar{\alpha})$:

$$\begin{tikzcd}
1 \arrow[d, "\varepsilon"'] \arrow[dr, bend left, "\bar{x}"] \arrow[drr, bend left, "\bar{x}"] & & \\
A^* \arrow[r, "r"] \arrow[rr, bend right, "r_{\bar{x}}"'] & \mathsf{free}(X, \alpha) \arrow[r, "i"] & X^n
\end{tikzcd}$$

Furthermore, we define the following set of equations:

$$\mathsf{Eq}(X, \alpha) = \mathsf{ker}(r)$$

where r is the reachability map for $(\mathsf{free}(X, \alpha), \bar{x}, \bar{\alpha})$. □

Note that

$$\mathsf{free}(X, \alpha) \cong A^* / \mathsf{Eq}(X, \alpha)$$

Definition 5.2 [cofree automaton, $\mathsf{coEq}(X, \alpha)$] Let $X = \{x_1, \ldots, x_n\}$ be the set of states of a finite automaton (X, α). We define a coloured automaton $\mathsf{cofree}(X, \alpha)$ in two steps, as follows:

(i) First we take the coproduct of the 2^n pointed automata (X, c, α) that we obtain by letting c range over the set $X \to 2$ of all colouring functions. This yields a coloured automaton $(\Sigma X, \hat{c}, \hat{\alpha})$ with

$$\Sigma X = \sum_{c:X \to 2} X_c$$

(where $X_c = X$), and with $\hat{c}$ and $\hat{\alpha}$ defined component-wise.

(ii) Next we define $(\mathsf{cofree}(X,\alpha), [\hat{c}], [\hat{\alpha}])$ by $\mathsf{cofree}(X,\alpha) = \Sigma X / \mathsf{ker}(o_{\hat{c}})$, where $o_{\hat{c}}$ is the observability map for $(\Sigma X, \hat{c}, \hat{\alpha})$:

$$\begin{tikzcd}
 & & 2 \\
\Sigma X \arrow[r, "q"] \arrow[urr, bend left, "\hat{c}"] \arrow[rr, bend right, "o_{\hat{c}}"'] & \mathsf{cofree}(X,\alpha) \arrow[r, "o"] \arrow[ur, bend left, "{[\hat{c}]}"] & 2^{A^*} \arrow[u, "\varepsilon?"']
\end{tikzcd}$$

and where $[\hat{c}]$ and $[\hat{\alpha}]$ are the extensions of $\hat{c}$ and $\hat{\alpha}$ to equivalence classes.

Furthermore, we define

$$\mathsf{coEq}(X,\alpha) = \mathsf{im}(o)$$

where o is the observability map for $(\mathsf{cofree}(X,\alpha), [\hat{c}], [\alpha])$. □

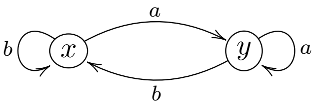
Note that

$$\mathsf{cofree}(X,\alpha) \cong \mathsf{coEq}(X,\alpha)$$

Theorem 5.3 *The set* $\mathsf{Eq}(X,\alpha)$ *is the* largest *set of equations satisfied by* (X,α). *The set* $\mathsf{coEq}(X,\alpha)$ *is the* smallest *set of coequations satisfied by* (X,α). □

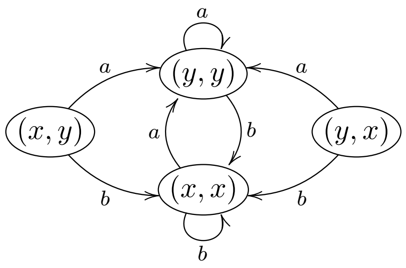
Example 5.4 [Example 4.3 continued] We consider our previous example

$$(Z,\gamma) \quad =$$



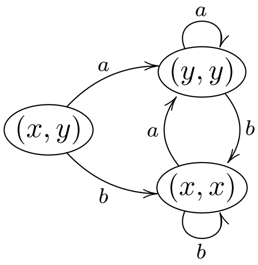
The product of (Z,x,γ) and (Z,y,γ) is:

$$(\Pi Z, (x,y), \bar{\gamma}) \quad =$$



Taking $\mathsf{im}(r_{(x,y)})$ yields the part that is reachable from (x,y):

$$(\mathsf{free}(Z,\gamma), (x,y), \bar{\gamma}) \quad =$$



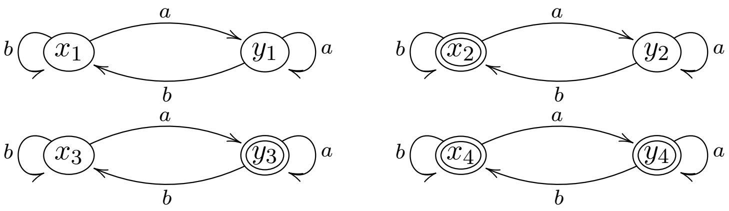
The set $\mathsf{Eq}(Z,\gamma)$ is defined as $\mathsf{ker}(r_{(x,y)})$, and consists of (the smallest bisimulation equivalence on (A^*,σ) generated by)

$$\mathsf{Eq}(Z,\gamma) = \quad \{aa = a,\, bb = b,\, ab = b,\, ba = a\}$$

This is the largest set of equations satisfied by (Z,γ).

Next we turn to coequations. The coproduct of all 4 coloured versions of (Z,γ) is

$(\Sigma Z, \hat{c}, \hat{\gamma}) \quad =$

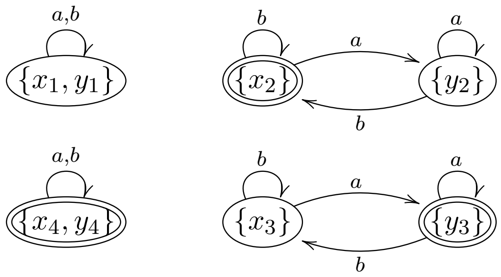


The observability map $o_{\hat{c}} : \Sigma Z \to 2^{A^*}$ is given by

$o_{\hat{c}}(x_1)$	$o_{\hat{c}}(y_1)$	$o_{\hat{c}}(x_2)$	$o_{\hat{c}}(y_2)$	$o_{\hat{c}}(x_3)$	$o_{\hat{c}}(y_3)$	$o_{\hat{c}}(x_4)$	$o_{\hat{c}}(y_4)$
$\emptyset$	$\emptyset$	$(a^*b)^*$	$(a^*b)^+$	$(b^*a)^+$	$(b^*a)^*$	A^*	A^*

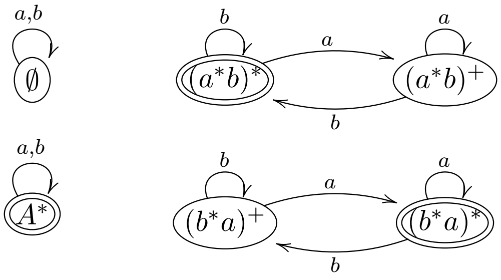
Computing the quotient $\Sigma Z/\mathsf{ker}(o_{\hat{c}})$ yields:

$(\mathsf{cofree}(Z,\gamma), [\hat{c}], [\hat{\gamma}]) =$



The image of this automaton under the reachability map $o : \mathsf{cofree}(Z,\gamma) \to 2^{A^*}$ is

$\mathsf{coEq}(Z,\gamma) =$



(3)

This is the smallest set of coequations satisfied by (Z,γ). □

Summarizing the present section, we have obtained, for every automaton (X,α), the following refinement of (2):

$$\begin{tikzcd}
1 \arrow[d, "\varepsilon"'] \arrow[rd, bend left, "\bar{x}"] \arrow[rrd, bend left, "x"] & & & & 2 \\
A^* \arrow[r, "r"'] \arrow[d, "\sigma"'] & \mathsf{free}(X,\alpha) \arrow[r] \arrow[d, "\bar{\alpha}"'] & X \arrow[r] \arrow[d, "\alpha"'] \arrow[rru, bend left, "c"] & \mathsf{cofree}(X,\alpha) \arrow[r, "o"'] \arrow[d, "{[\hat{\alpha}]}"] \arrow[ru, bend left, "{[\hat{c}]}"] & 2^{A^*} \arrow[u, "\varepsilon?"'] \arrow[d, "\tau"] \\
(A^*)^A \arrow[r] & \mathsf{free}(X,\alpha)^A \arrow[r] & X^A \arrow[r] & \mathsf{cofree}(X,\alpha)^A \arrow[r] & (2^{A^*})^A
\end{tikzcd}$$

where x ranges over the elements of X and c ranges over all possible colourings of X. The free and cofree automata represent the largest set of equations and the smallest set of coequations satisfied by (X, α):

$$\mathsf{Eq}(X,\alpha) = \mathsf{ker}(r) \qquad \mathsf{coEq}(X,\alpha) = \mathsf{im}(o)$$

Note that the free and cofree automata are constructed for the automaton (X, α), without point and without colouring. In conclusion, let us mention again that all of the above easily generalises to *infinite* X.

6 Varieties and covarieties

We define varieties and covarieties by means of equations and coequations, first for automata and next for languages.

Definition 6.1 [variety of automata] For every set E of equations we define the *variety* V_E by

$$V_E = \{\, (X,\alpha) \mid (X,\alpha) \models \ E \,\}$$

□

Definition 6.2 [covariety of automata] For every set D of coequations we define the *covariety* C_D by

$$C_D = \{\, (X,\alpha) \mid (X,\alpha) \models \ D \,\}$$

□

Every variety of automata defines a set of languages, which we will again call a variety. Dually, every covariety of automata defines a set of languages , which we will again call a covariety.

Definition 6.3 [variety and covariety of languages] Let V_E be a variety of automata. We define the *variety of languages* $L(V_E)$ by

$$L(V_E) = \{\, L \in 2^{A^*} \mid \langle L \rangle \in V_E \,\}$$

(where $\langle L \rangle$ is the subautomaton of $(2^{A^*}, \tau)$ generated by L). Dually, let C_D be a covariety of automata. We define the *covariety of languages* $L(C_D)$ by

$$L(C_D) = \{\, L \in 2^{A^*} \mid \langle L \rangle \in C \,\}$$

□

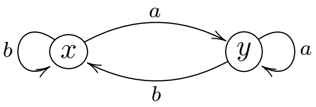
Proposition 6.4 *Every variety V_E is closed under the formation of* subautomata, homomorphic images, *and* products. □

Proposition 6.5 *Every covariety C_D is closed under the formation of* subautomata, homomorphic images, *and* coproducts. □

Proposition 6.6 *A covariety C_D is generally* not *closed under products.*

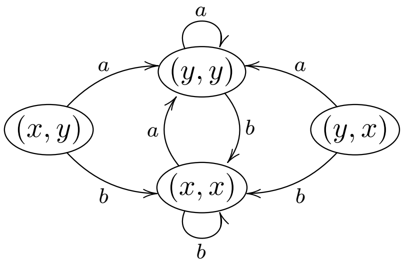
Proof. We give an example of a covariety that is not closed under products. We recall from Example 5.4 the automaton

$(Z, \gamma) \quad =$



We saw that $(Z,\gamma) \models D$, with $D = \mathsf{coEq}(Z,\gamma)$ as in (3). The product of (Z,γ) with itself is

$(Z^2, \bar{\gamma}) \quad =$



We define a colouring $c : Z^2 \to 2$ by

$c((x,y))$	$c((y,y))$	$c((x,x))$	$c((y,x))$
0	1	1	0

This colouring c induces the observability map $o_c : Z^2 \to 2^{A^*}$, given by

$o_c((x,y))$	$o_c((y,y))$	$o_c((x,x))$	$o_c((y,x))$
A^+	A^*	A^*	A^+

Because $A^+ \notin D$, the automaton $(Z^2, \bar{\gamma}) \not\models D$. Thus C_D is not closed under products. □

Corollary 6.7 Not *every covariety C_D is also a variety.* □

Here are some elementary properties of (co)equations and (covarieties).

Proposition 6.8 *For every set of equations $E \subseteq A^* \times A^*$,*

$$L(V_E) = \{ L \in 2^{A^*} \mid \forall (v,w) \in \tilde{E},\ L_v = L_w \}$$

where $\tilde{E}$ is the smallest congruence relation containing E. □

Theorem 6.9 (on equations and varieties) *Let $E \subseteq A^* \times A^*$ be a set of equations. The following statements are equivalent:*

0. *E is a congruence*
1. *$E = \mathsf{Eq}(X, \alpha)$ for some automaton (X, α)*
2. *$(A^*/E, [\sigma]) \models E$*
3. *$\mathsf{Eq}(A^*/E, [\sigma]) = E$*

(with σ as in (2)). Furthermore, any of the above implies:

4. *$L(V_E) = \{L \in 2^{A^*} \mid \forall (v, w) \in E,\ L_v = L_w\}$.*

□

Theorem 6.10 (on coequations and covarieties) *Let $D \subseteq 2^{A^*}$ be a set of coequations. The following statements are equivalent:*

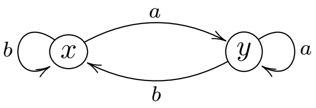
1. *$D = \mathsf{coEq}(X, \alpha)$ for some automaton (X, α)*
2. *$(D, \tau) \models D$*
3. *$\mathsf{coEq}(D, \tau) = D$*
4. *$L(C_D) = D$*

(with τ as in (2)). □

Corollary 6.11 *Every variety of languages $L(V_E)$ is also a covariety of languages.* □

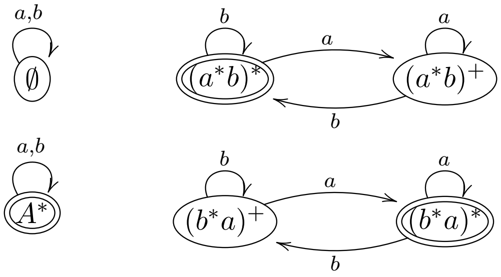
Example 6.12 [Example 5.4 continued] Recall the automaton

$(Z, \gamma) \quad =$



and recall

$\mathsf{coEq}(Z, \gamma) =$



The smallest *covariety* containing (Z, γ) is

$$C_{\mathsf{coEq}(Z,\gamma)}$$

It contains the languages

$$L(C_{\mathsf{coEq}(Z,\gamma)}) = \{\, \emptyset,\ (a^*b)^*,\ (a^*b)^+,\ (b^*a)^*,\ (b^*a)^+,\ A^* \,\}$$

The smallest *variety* containing (Z, γ) is

$$V_{\mathsf{Eq}(Z,\gamma)}$$

were we recall that $\mathsf{Eq}(Z,\gamma)$ is the smallest bisimulation equivalence (in fact, a congruence) generated by the set

$$\{aa = a,\, bb = b,\, ab = b,\, ba = a\}$$

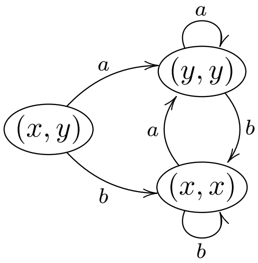
We have

$$\begin{aligned} L(V_{\mathsf{Eq}(Z,\gamma)}) &= \{L \in 2^{A^*} \mid (L_{aa} = L_a) \,\wedge\, (L_{bb} = L_b) \,\wedge\, (L_{ab} = L_b) \,\wedge\, (L_{ba} = L_a)\,\} \\ &= \{\,\emptyset, 1,\ (a^*b)^*,\ (a^*b)^+,\ 1 + (a^*b)^+,\ (b^*a)^*,\ (b^*a)^+,\ 1 + (b^*a)^+,\ A^+,\ A^*\,\} \end{aligned}$$

The latter set of languages can be, equivalently, determined using the fact that

$$\begin{aligned} V_{\mathsf{Eq}(Z,\gamma)} &= C_{\mathsf{coEq}(\,(A^*,\sigma)/\mathsf{Eq}(Z,\gamma)\,)} \\ &= C_{\mathsf{coEq}(\,\mathsf{free}(Z,\gamma)\,)} \end{aligned}$$

To this end, we recall that

$(\mathsf{free}(Z,\gamma), (x,y), \bar{\gamma}) \quad =$ 

and compute $\mathsf{coEq}(\,\mathsf{free}(Z,\gamma)\,)$ by means of the following table, which contains all possible colourings c of $\mathsf{free}(Z,\gamma)$, together with the corresponding value of o_c:

c	$c((x,y))$	$c((y,y))$	$c((x,x))$	$o_c((x,y))$	$o_c((y,y))$	$o_c((x,x))$
c_1	0	0	0	$\emptyset$	$\emptyset$	$\emptyset$
c_2	0	0	1	$(a^*b)^+$	$(a^*b)^+$	$(a^*b)^*$
c_3	0	1	0	$(b^*a)^+$	$(b^*a)^*$	$(b^*a)^+$
c_4	0	1	1	A^+	A^*	A^*
c_5	1	0	0	1	$\emptyset$	$\emptyset$
c_6	1	0	1	$1+(a^*b)^+$	$(a^*b)^+$	$(a^*b)^*$
c_7	1	1	0	$1+(b^*a)^+$	$(b^*a)^*$	$(b^*a)^+$
c_8	1	1	1	A^*	A^*	A^*

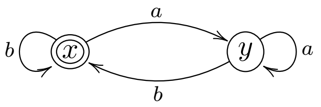
In the end, this leads to the same set of languages. We conclude this example by observing that

$$L(C_{\mathsf{coEq}(Z,\gamma)}) \subseteq L(V_{\mathsf{Eq}(Z,\gamma)})$$

as expected. □

Example 6.13 Here we focus on a single given language, say: $L = (a^*b)^*$. A minimal automaton for L is

$$(Z, x, c, \gamma) \;=$$



It follows from Example 6.12 that the smallest covariety of languages containing L is

$$L(C_{\mathsf{coEq}(Z,\gamma)}) = \{\, \emptyset,\, (a^*b)^*,\, (a^*b)^+,\, (b^*a)^*,\, (b^*a)^+,\, A^* \,\}$$

and that the smallest variety containing L is

$$L(V_{\mathsf{Eq}(Z,\gamma)}) = \{\, \emptyset, 1,\; (a^*b)^*,\, (a^*b)^+,\, 1{+}(a^*b)^+,\, (b^*a)^*,\, (b^*a)^+,\, 1{+}(b^*a)^+,\, A^+,\, A^* \,\}$$

□

Example 6.14 Here are some further examples of varieties and covarieties.

(i) The smallest congruence generated by $\{\, a = \varepsilon, b = \varepsilon \,\}$ is $E = A^* \times A^*$. As a consequence,

$$L(V_E) = \{\, \emptyset,\, A^* \,\}$$

The same for $E = \{\, b = \varepsilon, ab = \varepsilon, aa = a \,\}$.

(ii) If E is the smallest congruence generated by $\{aa = \varepsilon,\, b = \varepsilon\,\}$, then

$$L(V_E) = \{\, \emptyset,\, ((ab^*a) + b)^*,\, ((ab^*a) + b)^*ab^*,\, \{a, b\}^* \,\}$$

(iii) If E is the smallest congruence generated by $\{aa = \varepsilon,\, bb = \varepsilon\,\}$, then the variety $L(V_E)$ is infinite and contains both rational and non-rational languages.

(iv) For $D = 2^{A^*}$, the covariety C_D contains *all* automata (X, α).

(v) For $D = \mathsf{rat}(2^{A^*})$,

$$C_D = \{(X, \alpha) \mid (X, \alpha) \text{ is finitely generated}\,\}$$

that is, all (X, α) such that $\langle x \rangle \subseteq X$ is finite, for all $x \in X$.

(vi) If $D = \{\, \{a\},\, 1,\, \emptyset \,\}$ then $C_D = \emptyset$.

7 Transition monoids

For every (rational) language, one can construct its so-called *syntactic monoid* (that is, the transition monoid of its minimal automaton). Next every (classical, algebraic) variety V of monoids determines a class of languages L by the requirement that its syntactic monoid belongs to V. This is, in short, Eilenberg's definition of a variety of languages. In this section, we take a first step towards an understanding of the relation between Eilenberg's definition and the present one, by the observation that $\mathsf{free}(X, \alpha)$, for every automaton (X, α), is isomorphic to its transition monoid.

A *monoid* $(M, \cdot, 1)$ consists of a set M, a binary operation of multiplication that is associative, and a unit 1 with $m \cdot 1 = 1 \cdot m = m$. For every set, there is the monoid

$$(X^X, \cdot, 1_X)$$

defined by

$$X^X = \{\phi \mid \phi : X \to X\} \qquad 1_X(x) = x \qquad f \cdot g = g \circ f$$

Because of the isomorphism

$$X \to X^A \;\cong\; A \to X^X$$

we have for every automaton (X, α) and $a \in A$ a function

$$\tilde{a} : X \to X \qquad \tilde{a}(x) = \alpha(x)(a) = x_a$$

We use it to define for every automaton (X, α) a pointed automaton

$$(X^X, 1_X, \tilde{\alpha}) \qquad \tilde{\alpha}(\phi)(a) = \phi \cdot \tilde{a}$$

Next we define the *transition monoid* (cf. [18])

$$(\mathsf{trans}(X, \alpha), 1_X, \tilde{\alpha})$$

by $\mathsf{trans}(X, \alpha) = \mathsf{im}(r_{1_X})$, the image of the reachability map of $(X^X, 1_X, \tilde{\alpha})$:

$$\begin{tikzcd}
1 \arrow[d, "\varepsilon"'] \arrow[dr, bend left, "1_X"'] \arrow[drr, bend left, "1_X"] & & \\
A^* \arrow[r, "r"] \arrow[rr, bend right, "r_{1_X}"'] & \mathsf{trans}(X, \alpha) \arrow[r, "i"] & X^X
\end{tikzcd}$$

(where $r(a_1 \cdots a_n) = \tilde{a_1} \cdots \tilde{a_n}$, for $a_1 \cdots a_n \in A^*$).

Theorem 7.1 *For an automaton (X, α),*

$$(\mathsf{free}(X, \alpha), \bar{x}, \bar{\alpha}) \cong (\mathsf{trans}(X, \alpha), 1_X, \tilde{\alpha})$$

Proof. Let $X = \{x_1, \ldots, x_n\}$. For every $\bar{y} \in \mathsf{free}(X, \alpha)$ we define

$$\phi_{\bar{y}} : X \to X \qquad \phi_{\bar{y}}(x_i) = y_i$$

Then $\phi(\bar{y}) = \phi_{\bar{y}}$ defines an isomorphism of pointed automata. □

This elementary observation should form the basis for a detailed comparison of the present definition of variety of languages and Eilenberg's definition.

References

[1] M.A. Arbib and E.G. Manes. Adjoint machines, state-behaviour machines, and duality. *Journal of Pure and Applied Algebra*, 6:313–344, 1975.

[2] M.A. Arbib and H.P. Zeiger. On the relevance of abstract algebra to control theory. *Automatica*, 5:589–606, 1969.

[3] F. Bonchi, M. Bonsangue, H. Hansen, P. Panangaden, J. Rutten, and A. Silva. Algebra-coalgebra duality in Brzozowski's minimization algorithm. 2013. Submitted.

[4] A. Ballester-Bolinches, J.-E. Pin, and X. Soler-Escriva. Formations of finite monoids and formal languages: Eilenberg's variety theorem revisited. *Forum Mathematicum*, 2012.

[5] A. Ballester-Bolinches, J.-E. Pin, and X. Soler-Escriva. Languages associated with saturated formations of groups. *Forum Mathematicum*, 2013.

[6] F. Bonchi, M. Bonsangue, J. Rutten, and A. Silva. Brzozowski's algorithm (co)algebraically. In R. Constable and A. Silva, editors, *Logic and Program Semantics.*, volume 7230 of *LNCS*, pages 12–23, 2012.

[7] M. Bidoit, R. Hennicker, and A. Kurz. On the duality between observability and reachability. In Furio Honsell and Marino Miculan, editors, *FoSSaCS*, volume 2030 of *Lect. Notes in Comp. Sci.*, pages 72–87. Springer, 2001.

[8] J.A. Brzozowski. Derivatives of regular expressions. *Journal of the ACM*, 11(4):481–494, 1964.

[9] C. Cirstea. On specification logics for algebra-coalgebra structures: Reconciling reachability and observability. In *Proceedings FoSSaCS*, pages 82–97, 2002.

[10] J.H. Conway. *Regular algebra and finite machines.* Chapman and Hall, 1971.

[11] S. Eilenberg. *Automata, languages and machines (Vol. A).* Pure and applied mathematics. Academic Press, 1974.

[12] S. Eilenberg. *Automata, languages and machines (Vol. B).* Pure and applied mathematics. Academic Press, 1976.

[13] M. Gehrke, S. Grigorieff, and J.-E. Pin. Duality and equational theory of regular languages. In *Proceedings ICALP*, volume 5126 of *LNCS*, pages 246–257, 2008.

[14] R. Kalman. On the general theory of control systems. *IRE Transactions on Automatic Control*, 4(3):110–110, 1959.

[15] R. E. Kalman, P. L. Falb, and M. A. Arbib. *Topics in Mathematical Systems Theory.* McGraw Hill, 1969.

[16] S.C. Kleene. Representation of events in nerve nets and finite automata. In Shannon and McCarthy, editors, *Automata Studies*, pages 3–41. Princeton Univ. Press, 1956.

[17] E.G. Manes and M.A. Arbib. *Algebraic approaches to program semantics.* Texts and monographs in computer science. Springer-Verlag, 1986.

[18] J.-E. Pin. Syntactic semigroups. *Handbook of language theory, Vol. I*, pages 679–746, 1997.

[19] J.J.M.M. Rutten. Automata and coinduction (an exercise in coalgebra). In D. Sangiorgi and R. de Simone, editors, *Proceedings of CONCUR'98*, volume 1466 of *LNCS*, pages 194–218, 1998.

[20] J.J.M.M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249(1):3–80, 2000. Fundamental Study.

Layout Randomization and Nondeterminism

Martín Abadi

Microsoft Research, Silicon Valley
and UC Santa Cruz

Jérémy Planul

Stanford University

Gordon Plotkin

LFCS, Informatics, University of Edinburgh
and Microsoft Research, Silicon Valley

Abstract

In security, layout randomization is a popular, effective attack mitigation technique. Recent work has aimed to explain it rigorously, focusing on deterministic systems. In this paper, we study layout randomization in the presence of nondeterministic choice. We develop a semantic approach based on denotational models and simulation relations. This approach abstracts from language details, and helps manage the delicate interaction between probabilities and nondeterminism.

Keywords: security, semantics, probabilities, nondeterminism, full abstraction.

1 Introduction

Randomization has important applications in security, ranging from probabilistic cryptographic schemes [10] to the introduction of artificial diversity in low-level software protection [8]. Developing rigorous models and analyses of the systems that employ randomization can be challenging, not only because of the intrinsic difficulty of reasoning about probabilities but also because these systems typically exhibit many other interesting features. Some of these features, such as assumed bounds on the capabilities and the computational complexity of attackers, stem directly from security considerations. Others, such as nondeterminism, need not be specifically related to security, but arise because of the generality of the ambient computational models, which may for example include nondeterministic scheduling for concurrent programs and for network protocols.

The form of randomization that we explore in this paper is layout randomization in software systems (e.g., [6,18,7]). Layout randomization refers to a body of widely

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.com/locate/entcs

used techniques that place data and code randomly in memory. In practice, these techniques effectively thwart many attacks that assume knowledge of the location of data and code. Recent research by the authors and others aims to develop rigorous models and proofs for layout randomization [19,3,13,2]. The research to date has focused on deterministic, sequential programs. Here, we consider layout randomization for programs that may make nondeterministic choices.

We phrase our study in terms of a high-level language in which variables are abstract (symbolic) locations, and a low-level language in which they are mapped to random natural-number addresses in memory. Both languages include a standard construct for nondeterministic choice. We give models for the languages. For each language, we also define a contextual implementation relation. Intuitively, a context may represent an attacker, so contextual implementation relations may serve, in particular, for expressing standard security properties. We characterize contextual implementation relations in terms of semantic simulation relations (so-called logical relations). Throughout, the low-level relations are probabilistic. Via the simulation relations, we obtain a semantic correspondence between the high-level and low-level worlds. Basically, simulation relations in one world induce simulation relations in the other, and therefore contextual implementation in one world implies contextual implementation in the other.

Thus, our approach emphasizes semantic constructions. In comparison with prior syntactic work, arguments via models arguably lead to more satisfying security arguments, independent of superficial details of particular languages (as layout randomization is largely language-agnostic in practice). They also help reconcile probabilities and nondeterminism, which have a rich but thorny interaction.

Some of the difficulties of this interaction have been noticed in the past. For instance, in their development of a framework for the analysis of security protocols [15, Section 2.7], Lincoln et al. observed:

> our intention is to design a language of communicating processes so that an adversary expressed by a set of processes is restricted to probabilistic polynomial time. However, if we interpret parallel composition in the standard nondeterministic fashion, then a pair of processes may nondeterministically "guess" any secret information.

They concluded:

> Therefore, although nondeterminism is a useful modeling assumption in studying correctness of concurrent programs, it does not seem helpful for analyzing cryptographic protocols.

Thus, they adopted a form of probabilistic scheduling, and excluded nondeterminism. In further work, Mitchell et al. [17] refined the framework, in particular defining protocol executions by reference to any polynomial-time probabilistic scheduler that operates uniformly over certain kinds of choices. The uniformity prevents collusion between the scheduler and an attacker. Similarly, Canetti et al. [4] resolved nondeterminism by task schedulers, which do not depend on dynamic information generated during probabilistic executions; they thus generated sets of trace distributions, one for each task schedule.

From a semantic perspective, a nondeterministic program denotes a function that produces a set of possible outcomes; equally, a probabilistic program represents a function that produces a distribution over outcomes. Rigorous versions of these statements can be cast in terms of powerdomains and probabilistic powerdomains [9]. In principle, a nondeterministic and probabilistic program may represent either a function producing a set of distributions over outcomes or else one producing a distribution over sets of outcomes. However it seems that only the former option, where nondeterministic choice is resolved before probabilistic choice, leads to a satisfactory theory if, for example, one wishes to retain all the usual laws for both forms of nondeterminism [16,21,11].

To illustrate these options, imagine a two-player game in which Player I chooses a bit b_{I} at random, Player II chooses a bit b_{II} nondeterministically, and Player I wins if and only if $b_{\mathrm{I}} = b_{\mathrm{II}}$. The system composed of the two players may be seen as producing a set of distributions or a distribution on sets of outcomes.

- With the former view, we can say that, in each possible distribution, Player I wins with probability 1/2.
- On the other hand, with the latter view, we can say only that, with probability 1, Player I may win and may lose.

The former view is preferable in a variety of security applications, in which we may wish to say that no matter what an attacker does, or how nondeterministic choices are resolved, some expected property holds with high probability.

However, in our work, it does not suffice to resolve nondeterministic choice before probabilistic choice, as we explain in detail below, fundamentally because the probabilistic choices that we treat need not be independent. Instead, we construct a more sophisticated model that employs random variables, here maps from memory layouts to outcomes. The memory layouts form the sample space of the random variables, and, as usual, one works relative to a given distribution over the sample space.

Beyond the study of layout randomization, it seems plausible that an approach analogous to ours could be helpful elsewhere in security analysis. Our models may also be of interest on general grounds, as a contribution to a long line of research on programming-language semantics for languages with nondeterministic and probabilistic choice. Specifically, the models support a treatment of dependent probabilistic choice combined with nondeterminism, which as far as we know has not been addressed in the literature. Finally, the treatment of contextual implementation relations and simulation relations belongs in a long line of research on refinement.

Contents

In Section 2 we review some preliminary material on cpos.

In Section 3, we consider a high-level language, with abstract locations, standard imperative constructs, and nondeterminism, and describe its denotational and operational semantics. We define a contextual implementation relation with respect to contexts that represent attackers, which we call public contexts; for this purpose, we distinguish public locations, which attackers can access directly, from private locations. We also define a simulation relation, and prove that it coincides with the contextual implementation relation. The main appeal of the simulation relation, as usual, is that it does not require reasoning about all possible contexts.

In Section 4, we similarly develop a lower-level language in which programs may use natural-number memory addresses (rather than abstract locations). Again, we define a denotational semantics, an operational semantics, a contextual implementation relation, and a simulation relation. These definitions are considerably more delicate than those of the high-level language, in particular because they refer to layouts, which map abstract locations to concrete natural-number addresses, and which may be chosen randomly (so we often make probabilistic statements).

In Section 5, we relate the high-level and the low-level languages. We define a simple compilation function that maps from the former to the latter. We then establish that if two high-level commands are in the contextual implementation relation, then their low-level counterparts are also in the contextual implementation relation. The proof leverages simulation relations. In semantics parlance, this result is a full-abstraction theorem; the use of public contexts that represent attackers, however, is motivated by security considerations, and enable us to interpret this theorem as providing a formal security guarantee for the compilation function, modulo a suitable random choice of memory layouts.

Finally, in Section 6 we conclude by discussing some related and further work.

2 Preliminaries on cpos

We take a cpo to be a partial order P closed under increasing ω-sups, and consider sets to be cpos with the discrete ordering. We write $P_\bot$ for the lift of P, viz. P extended by the addition of a least element, $\bot$. Products $P \times Q$ and function spaces $P \to Q$ (which we may also write as Q^P) are defined as usual, with the function space consisting of all continuous functions (those monotonic functions preserving the ω-lubs).

We use the lower, or Hoare, powerdomain $\mathcal{H}(P)$ of the nonempty, downwards, and ω-sup-closed subsets of P, ordered by inclusion. The lower powerdomain is the simplest of the three powerdomains, and models “may” or “angelic” nondeterminism; the others (upper and convex) may also be worth investigating.

For any nonempty $X \subseteq P$, we write $X\!\downarrow$ for the downwards closure $\{y \mid \exists x \in X.\, y \leq x\}$ of X. We also write X^* for the downwards and ω-sup closure of X (which is typically the same as $X\!\downarrow$ in the instances that arise below).

Both $\mathcal{H}(-)$ and $\mathcal{H}(-_\bot)$ are monads (those for lower nondeterminism, and lower nondeterminism and nontermination, respectively). The unit of the former is $x \mapsto \{x\}\!\downarrow$ and any continuous map $f : P \to \mathcal{H}(Q)$ has an extension $f^\dagger : \mathcal{H}(P) \to \mathcal{H}(Q)$ given by:

$$f^\dagger(X) = (\bigcup_{x \in X} f(x))^*$$

For the latter the unit is $x \mapsto \{x\}\!\downarrow$ and the extension $f^\dagger : \mathcal{H}(P_\bot) \to \mathcal{H}(Q_\bot)$ of a continuous map $f : P \to \mathcal{H}(Q_\bot)$ is given by:

$$f^\dagger(X) = \{\bot\} \cup (\bigcup_{x \in X \backslash \{\bot\}} f(x))^*$$

3 The high-level language

In this section, we define our high-level language. In this language, locations are symbolic names, and we use an abstract store to link those locations to their contents, which are natural numbers.

For simplicity, the language lacks data structures and higher-order features. Therefore, locations cannot contain arrays or functions (cf. [2]), except perhaps through encodings. So the language does not provide a direct model of overflows and code-injection attacks, for instance.

There are many other respects in which our languages and their semantics are not maximally expressive, realistic, and complex. They are however convenient for our study of nondeterminism and of the semantic approach to layout randomization.

3.1 Syntax and informal semantics

The syntax of the high-level language includes categories for natural-number expressions, boolean expressions, and commands:

$$
\begin{aligned}
e &::= k \mid !l_{\texttt{loc}} \mid e + e \mid e * e \\
b &::= e \leq e \mid \neg b \mid \mathtt{tt} \mid \mathtt{ff} \mid b \vee b \mid b \wedge b \\
c &::= l_{\texttt{loc}} := e \mid \texttt{if}\, b\, \texttt{then}\, c\, \texttt{else}\, c \mid \texttt{skip} \mid c; c \mid c + c \mid \texttt{while}\, b\, \texttt{do}\, c
\end{aligned}
$$

where k ranges over numerals, and l over a given finite set of store locations Loc. Natural-number expressions are numerals, dereferencing of memory locations, sums, or products. Boolean expressions are inequalities on natural-number expressions, negations, booleans, disjunctions, or conjunctions. Commands are assignments at a location, conditionals, skip, sequences, nondeterministic choices, or loops. Command contexts $C[\ ]$ are commands with holes; we write $C[c]$ for the command obtained by filling all the holes in $C[\ ]$ with c. We further use trivial extensions of this language, in particular with additional boolean and arithmetic expressions.

We assume that the set of store locations Loc is the union of two disjoint sets of locations PubLoc (public locations) and PriLoc (private locations). Let c be a command or a command context. We say that c is public if it does not contain any occurrence of $l_{\texttt{loc}} := v$ or $!l_{\texttt{loc}}$ for $l \in \mathrm{PriLoc}$. As in previous work [3], we model attackers by such public commands and command contexts; thus, attackers have direct access to public locations but not, by default, to private locations.

The distinction between public and private locations is directly analogous to that between external and internal state components in automata and other specification formalisms (e.g., [1]). It also resembles distinctions in information-flow systems, which often categorize variables into levels (e.g., [20]), and typically aim to prevent flows of information from “high” to “low” levels. We do not impose any such information-flow constraint: we permit arbitrary patterns of use of public and private locations. Nevertheless, we sometimes use h for a private location and l for a public location, and also associate the symbols H and L with private and public locations, respectively.

$$\begin{array}{llll}
\llbracket l_{\texttt{loc}} := e \rrbracket(s) & = \eta(s[l \mapsto \llbracket e \rrbracket(s)]) & \llbracket \mathsf{skip} \rrbracket(s) & = \eta(s) \\
\llbracket \texttt{if}\, b\, \texttt{then}\, c\, \texttt{else}\, c' \rrbracket(s) & = \begin{cases} \llbracket c \rrbracket(s) & \llbracket b \rrbracket(s) = \mathtt{tt} \\ \llbracket c' \rrbracket(s) & \llbracket b \rrbracket(s) = \mathtt{ff} \end{cases} & \begin{array}{l} \llbracket c; c' \rrbracket(s) \\ \llbracket c + c' \rrbracket(s) \end{array} & \begin{array}{l} = \llbracket c' \rrbracket^{\dagger}(\llbracket c \rrbracket(s)) \\ = \llbracket c \rrbracket(s) \cup \llbracket c' \rrbracket(s) \end{array} \\
\llbracket \texttt{while}\, b\, \texttt{do}\, c \rrbracket & \multicolumn{3}{l}{= \mu\, \theta : S \to \mathcal{H}(S_{\perp}).\, \lambda s : S.\, \begin{cases} \eta(s) & (\llbracket b \rrbracket(s) = \mathtt{ff}) \\ \theta^{\dagger}(\llbracket c \rrbracket(s)) & (\llbracket b \rrbracket(s) = \mathtt{tt}) \end{cases}}
\end{array}$$

Fig. 1. High-level denotational semantics

3.2 Denotational semantics

A store s is a function from a finite set Loc of store locations to natural numbers. When Loc consists of h and l, for example, we write $(h \mapsto m, l \mapsto n)$ for the store that maps h to m and l to n. A public (private) store is a function from PubLoc (PriLoc) to natural numbers. We write S for the set of stores, S_L for the set of public stores, and S_H for the set of private stores. Note the natural functions:

$$\begin{tikzcd} S_L & S \arrow[l, "L"'] \arrow[r, "H"] & S_H \end{tikzcd}$$

We write s_L for $L(s)$ and $s =_L s'$ when $s_L = s'_L$, and similarly for H.

The denotational semantics

$$\llbracket e \rrbracket : \text{Store} \to \mathbb{N} \qquad \llbracket b \rrbracket : \text{Store} \to \mathbb{B}$$

of expressions are defined in the standard way with, in particular, $\llbracket !l_{\texttt{loc}} \rrbracket(s) = s(l)$. The denotational semantics

$$\llbracket c \rrbracket : S \to \mathcal{H}(S_{\perp})$$

of commands is given in Figure 1, where the semantics of the while loop is the standard least-fixed point one.

Example 3.1 Consider the two commands:

$$c_0 = (h := \mathtt{tt}; l := \neg !l) + (h := \mathtt{ff}) \qquad c_1 = (h := \mathtt{tt}; l := \mathtt{tt}) + (h := \mathtt{ff}; l := \mathtt{ff})$$

According to the semantics, $\llbracket c_0 \rrbracket$ maps any store where l is $\mathtt{tt}$ to $\{(h \mapsto \mathtt{tt}, l \mapsto \mathtt{ff}), (h \mapsto \mathtt{ff}, l \mapsto \mathtt{tt})\}{\downarrow}$, and any store where l is $\mathtt{ff}$ to $\{(h \mapsto \mathtt{tt}, l \mapsto \mathtt{tt}), (h \mapsto \mathtt{ff}, l \mapsto \mathtt{ff})\}{\downarrow}$, while $\llbracket c_1 \rrbracket$ maps any store to $\{(h \mapsto \mathtt{tt}, l \mapsto \mathtt{tt}), (h \mapsto \mathtt{ff}, l \mapsto \mathtt{ff})\}{\downarrow}$. In sum, we may write:

$$\begin{aligned}
\llbracket c_0 \rrbracket(h \mapsto \_, l \mapsto \mathtt{tt}) &= \{(h \mapsto \mathtt{tt}, l \mapsto \mathtt{ff}), (h \mapsto \mathtt{ff}, l \mapsto \mathtt{tt})\}{\downarrow} \\
\llbracket c_0 \rrbracket(h \mapsto \_, l \mapsto \mathtt{ff}) &= \{(h \mapsto \mathtt{tt}, l \mapsto \mathtt{tt}), (h \mapsto \mathtt{ff}, l \mapsto \mathtt{ff})\}{\downarrow} \\
\llbracket c_1 \rrbracket(h \mapsto \_, l \mapsto \_) &= \{(h \mapsto \mathtt{tt}, l \mapsto \mathtt{tt}), (h \mapsto \mathtt{ff}, l \mapsto \mathtt{ff})\}{\downarrow}
\end{aligned}$$

Note that the semantics of the two commands are different. Nevertheless, below we show that these two commands are in a sense equivalent (with respect to public contexts). □

$$\langle l_{\texttt{loc}} := e, s\rangle \to s[l \mapsto [\![e]\!]_s] \qquad \frac{[\![b]\!]_s = \mathtt{tt}}{\langle \texttt{if}\, b\, \texttt{then}\, c\, \texttt{else}\, c', s\rangle \to \langle c, s\rangle}$$

$$\frac{[\![b]\!]_s = \mathtt{ff}}{\langle \texttt{if}\, b\, \texttt{then}\, c\, \texttt{else}\, c', s\rangle \to \langle c', s\rangle} \qquad \langle \mathsf{skip}, s\rangle \to s \qquad \frac{\langle c, s\rangle \to \langle c', s'\rangle}{\langle c; c'', s\rangle \to \langle c'; c'', s'\rangle}$$

$$\frac{\langle c, s\rangle \to s'}{\langle c; c'', s\rangle \to \langle c'', s'\rangle} \qquad \langle c + c', s\rangle \to \langle c, s\rangle \qquad \langle c + c', s\rangle \to \langle c', s\rangle$$

$$\frac{[\![b]\!]_s = \mathtt{ff}}{\langle \texttt{while}\, b\, \texttt{do}\, c, s\rangle \to s} \qquad \frac{[\![b]\!]_s = \mathtt{tt}}{\langle \texttt{while}\, b\, \texttt{do}\, c, s\rangle \to \langle c; \texttt{while}\, b\, \texttt{do}\, c, s\rangle}$$

Fig. 2. High-level operational semantics

3.3 Operational semantics

The high-level language has a straightforward small-step operational semantics. In this semantics, a high-level state is a pair $\langle c, s\rangle$ of a command and a store or, in case of termination, just a store s. The transition relation $\to$ is a binary relation on such states. Figure 2 gives the rules for $\to$.

Proposition 3.2 (Operational/denotational consistency) *Let c be a command and s be a store. We have*

$$[\![c]\!](s) = \{s' | \langle c, s\rangle \to^* s'\} \cup \bot$$

3.4 Implementation relations and equivalences

3.4.1 Contextual pre-order

We introduce a contextual pre-order $\sqsubseteq_L$ on commands. Intuitively, $c \sqsubseteq_L c'$ may be interpreted as saying that c "refines" (or "implements") c', in the sense that the publicly observable outcomes that c can produce are a subset of those that c' permits, in every public context and from every initial store. Thus, let $f = [\![C[c]]\!]$ and $f' = [\![C[c']]\!]$ for an arbitrary public context C, and let s_0 be a store; then for every store s in $f(s_0)$ there is a store s' in $f'(s_0)$ that coincides with s on public locations. Note that we both restrict attention to public contexts and compare s and s' only on public locations.

We define $\sqsubseteq_L$ and some auxiliary relations as follows:

- For $X \in \mathcal{H}(S_\bot)$, we set:

$$X_L = \{s_L \mid s \in X\} \cup \{\bot\}$$

- For $f, f' : S \to \mathcal{H}(S_\bot)$, we write that $f \leq_L f'$ when, for every store s_0, we have $f(s_0)_L \leq f'(s_0)_L$.
- Let c and c' be two commands. We write that $c \sqsubseteq_L c'$ when, for every public command context C, we have $[\![C[c]]\!] \leq_L [\![C[c']]\!]$.

Straightforwardly, this contextual pre-order relation yields a notion of contextual equivalence with respect to public contexts.

3.4.2 Simulation

In addition to a contextual pre-order, we introduce a simulation relation $\preceq$ whose main advantage, as usual, is that it does not require reasoning about contexts.

As in much previous work, one might expect a simulation relation between two commands c and c' to be a relation on stores that respects the observable parts of these stores, and such that if s_0 is related to s_1 and c can go from s_0 to s'_0 then there exists s'_1 such that s'_0 is related to s'_1 and c' can go from s_1 to s'_1. In our setting, respecting the observable parts of stores means that related stores give the same values to public locations (much like refinement mappings preserve externally visible state components [1], and low-bisimulations require equivalence on low-security variables [20]).

Although this idea could lead to a sound proof technique for the contextual pre-order, it does not suffice for completeness. Indeed, forward simulations, of the kind just described, are typically incomplete on their own for nondeterministic systems. They can be complemented with techniques such as backward simulation, or generalized (e.g., [1,14,5]).

Here we develop one such generalization. Specifically, we use relations on sets of stores. We build them from relations over $\mathcal{H}(S_{H\perp})$ as a way of ensuring the condition that public locations have the same values, mentioned above. We also require other standard closure conditions. Our relations are similar to the ND measures of Klarlund and Schneider [14]. Their work takes place in an automata-theoretic setting; automata consist of states (which, intuitively, are private) and of transitions between those states, labeled by events (which, intuitively, are public). ND measures are mappings from states to sets of finite sets of states, so can be seen as relations between states and finite sets of states. The finiteness requirement, which we do not need, allows a fine-grained treatment of infinite execution paths via König's Lemma.

First, we extend relations R over $\mathcal{H}(S_{H\perp})$ to relations R^+ over $\mathcal{H}(S_\perp)$, as follows. For any $X \in \mathcal{H}(S_\perp)$ and $s \in S_L$, we define $X_s \in \mathcal{H}(S_{H\perp})$ by:

$$X_s = \{s'_H \mid s' \in X, s'_L = s\} \cup \{\perp\}$$

and then we define R^+ by:

$$XR^+Y \equiv_{\text{def}} \forall s \in S_L. \, (X_s \neq \{\perp\} \Rightarrow Y_s \neq \{\perp\}) \wedge X_s R Y_s$$

If R is reflexive (respectively, is closed under increasing ω-sups; is right-closed under $\leq$ and closed under binary unions) the same holds for R^+. Also, if XR^+Y then $X_L \leq Y_L$.

For any $f, f' : S_\perp \to \mathcal{H}(S_\perp)$ and relation R over $\mathcal{H}(S_{H\perp})$ we write that $f \preceq_R f'$ when:

$$\forall X, Y \in \mathcal{H}(S_\perp). \, XR^+Y \Rightarrow f^\dagger(X)R^+f'^\dagger(Y)$$

Finally, we write that $f \preceq f'$ if $f \preceq_R f'$ for some reflexive R closed under increasing ω-sups, right-closed under $\leq$, and closed under binary unions.

3.4.3 Contextual pre-order vs. simulation

The contextual pre-order coincides with the simulation relation:

Theorem 3.3 *Let c and c' be two commands of the high-level language. Then $c \sqsubseteq_L c'$ if and only if $[\![c]\!] \preceq [\![c']\!]$.*

Example 3.4 We can verify that c_0 and c_1, introduced in Example 3.1, are equivalent (with R the full relation). For instance, let $S_0 = \{(h \mapsto \mathtt{ff}, l \mapsto \mathtt{tt})\}\downarrow$ and $S_1 = \{(h \mapsto \mathtt{tt}, l \mapsto \mathtt{tt})\}\downarrow$. We have $S_0 R^+ S_1$, and:

$$\begin{aligned} [\![c_0]\!]^\dagger(S_0) &= \{(h \mapsto \mathtt{tt}, l \mapsto \mathtt{ff}), (h \mapsto \mathtt{ff}, l \mapsto \mathtt{tt})\}\downarrow \\ [\![c_1]\!]^\dagger(S_1) &= \{(h \mapsto \mathtt{tt}, l \mapsto \mathtt{tt}), (h \mapsto \mathtt{ff}, l \mapsto \mathtt{ff})\}\downarrow \end{aligned}$$

We can then check that:

$$[\![c_0]\!]^\dagger(S_0) R^+ [\![c_1]\!]^\dagger(S_1)$$

□

Example 3.5 In this example, we study the two commands

$$\begin{aligned} c_2 &= \texttt{if}\, h = 0 \,\texttt{then}\, l := 1 \,\texttt{else}\, (h := 0) + (h :=!h - 1) \\ c_3 &= \texttt{if}\, h = 0 \,\texttt{then}\, l := 1 \,\texttt{else}\, (h := 0) + \mathsf{skip} \end{aligned}$$

which seem to share the same behavior on public variables, but that are inherently different because of their behavior on private variables. According to the semantics, we have:

$$\begin{aligned} [\![c_2]\!](h \mapsto 0, l \mapsto \_) &= \{(h \mapsto 0, l \mapsto 1)\}\downarrow \\ [\![c_2]\!](h \mapsto j+1, l \mapsto k) &= \{(h \mapsto j, l \mapsto k), (h \mapsto 0, l \mapsto k)\}\downarrow \\ [\![c_3]\!](h \mapsto 0, l \mapsto \_) &= \{(h \mapsto 0, l \mapsto 1)\}\downarrow \\ [\![c_3]\!](h \mapsto j+1, l \mapsto k) &= \{(h \mapsto j+1, l \mapsto k), (h \mapsto 0, l \mapsto k)\}\downarrow \end{aligned}$$

We can verify that $c_2 \preceq_R c_3$, with R defined as the smallest relation that satisfies our conditions (reflexivity, etc.) and such that

$$\{(h \mapsto k)\} R \{(h \mapsto k')\} \qquad \text{for all } k \leq k'$$

For instance, suppose that $S_0 = \{(h \mapsto 5, l \mapsto 0)\}\downarrow$ and that $S_1 = \{(h \mapsto 7, l \mapsto 0)\}\downarrow$. We have $S_0 R^+ S_1$, and:

$$\begin{aligned} [\![c_2]\!]^\dagger(S_0) &= \{(h \mapsto 4, l \mapsto 0), (h \mapsto 0, l \mapsto 0)\}\downarrow \\ [\![c_3]\!]^\dagger(S_1) &= \{(h \mapsto 7, l \mapsto 0), (h \mapsto 0, l \mapsto 0)\}\downarrow \end{aligned}$$

We can then check that:

$$[\![c_2]\!]^\dagger(S_0) R^+ [\![c_3]\!]^\dagger(S_1)$$

On the other hand, there is no suitable relation R such that $c_3 \preceq_R c_2$. If there were such a relation R, it would have to be reflexive, so $\{(h \mapsto 1)\}\ R\ \{(h \mapsto 1)\}$. Suppose that $S_0 = \{(h \mapsto 1, l \mapsto 0)\}{\downarrow}$ and that $S_1 = \{(h \mapsto 1, l \mapsto 0)\}{\downarrow}$. We have $S_0 R^+ S_1$, and:

$$\begin{aligned} [\![c_3]\!]^\dagger(S_0) &= \{(h \mapsto 1, l \mapsto 0), (h \mapsto 0, l \mapsto 0)\}{\downarrow} \\ [\![c_2]\!]^\dagger(S_1) &= \{(h \mapsto 0, l \mapsto 0)\}{\downarrow} \end{aligned}$$

We need

$$\{(h \mapsto 1, l \mapsto 0), (h \mapsto 0, l \mapsto 0)\}{\downarrow}\ R^+ \{(h \mapsto 0, l \mapsto 0)\}{\downarrow}$$

hence $\{(h \mapsto 1)\} R \{(h \mapsto 0)\}$. Let $S_2 = \{(h \mapsto 1, l \mapsto 0)\}{\downarrow}$ and $S_3 = \{(h \mapsto 0, l \mapsto 0)\}{\downarrow}$. We have $S_2 R^+ S_3$, and:

$$\begin{aligned} [\![c_3]\!]^\dagger(S_2) &= \{(h \mapsto 1, l \mapsto 0), (h \mapsto 0, l \mapsto 0)\}{\downarrow} \\ [\![c_2]\!]^\dagger(S_3) &= \{(h \mapsto 0, l \mapsto 1)\}{\downarrow} \end{aligned}$$

Since the values of l do not match, we cannot have $[\![c_3]\!]^\dagger(S_2) R^+ [\![c_2]\!]^\dagger(S_3)$, hence $c_3 \not\preceq_R c_2$.

As predicted by Theorem 3.3, we also have $c_3 \not\sqsubseteq_L c_2$. Indeed, for $C = \_\,;\_$ and $s_0 = (h \mapsto 1, l \mapsto 0)$, we have $[\![C[c_3]]\!](s_0) \not\leq_L [\![C[c_2]]\!](s_0)$. □

4 The low-level language

In this section, we define our low-level language. In this language, we use concrete natural-number addresses for memory. We still use abstract location names, but those are interpreted as natural numbers (according to a memory layout), and can appear in arithmetic expressions.

4.1 Syntax and informal semantics

The syntax of the low-level language includes categories for natural-number expressions, boolean expressions, and commands:

$$\begin{aligned} e &::= k \mid l_{\texttt{nat}} \mid !e \mid e + e \mid e * e \\ b &::= e \leq e \mid \neg b \mid \mathrm{tt} \mid \mathrm{ff} \mid b \vee b \mid b \wedge b \\ c &::= e := e \mid \texttt{if}\, b\, \texttt{then}\, c\, \texttt{else}\, c \mid \texttt{skip} \mid c; c \mid c + c \mid \texttt{while}\, b\, \texttt{do}\, c \end{aligned}$$

where k ranges over numerals, and l over the finite set of store locations. Boolean expressions are as in the high-level language. Natural-number expressions and commands are also as in the high-level language, except for the inclusion of memory locations among the natural-number expressions, and for the dereferencing construct $!e$ and assignment construct $e := e'$ where e is an arbitrary natural-number expression (not necessarily a location).

Importantly, memory addresses are natural numbers, and a memory is a partial function from those addresses to contents. We assume that accessing an address at which the memory is undefined constitutes an error that stops execution immediately. In this respect, our language relies on the "fatal-error model" of Abadi and Plotkin [3]. With more work, it may be viable to treat also the alternative "recoverable-error model", which permits attacks to continue after such accesses, and therefore requires a bound on the number of such accesses.

4.2 Denotational semantics

4.2.1 Low-level memories, layouts, and errors

We assume given a natural number $\mathrm{r} > 0$ that specifies the size of the memory. A memory m is a partial function from $\{1, \ldots, \mathrm{r}\}$ to natural numbers; we write Mem for the set of memories. A memory layout w is an injection from Loc to $\{1, \ldots, \mathrm{r}\}$. We consider only memory layouts that extend a given public memory layout w_p (an injection from PubLoc to $\{1, \ldots, \mathrm{r}\}$), fixed in the remaining of the paper. We let W be the set of those layouts.

The security of layout randomization depends on the randomization itself. We let d be a probability distribution on memory layouts (that extend w_p). When φ is a predicate on memory layouts, we write $P_d(\varphi(w))$ for the probability that $\varphi(w)$ holds with w sampled according to d.

Given a distribution d on layouts, we write δ_d for the minimum probability for a memory address to have no antecedent location (much as in [3]):

$$\delta_d = \min_{i \in \{1, \ldots, \mathrm{r}\} \setminus \mathrm{ran}(w_p)} P_d(i \notin \mathrm{ran}(w))$$

This probability also bounds 1 minus the maximum probability for an adversary to guess a location. For common distributions (e.g., the uniform distribution), δ_d approaches 1 as r grows, indicating that adversaries fail most of the time. We assume d fixed below, and may omit it, writing δ for δ_d.

The denotational semantics of the low-level language uses the "error + nontermination" monad $P_{\xi\perp} =_{\mathrm{def}} (P + \{\xi\})_\perp$, which first adds an "error" element ξ to P and then a least element. As the monad is strong, functions $f\colon P_1 \times \ldots \times P_n \to Q_{\xi\perp}$ extend to functions $\overline{f}$ on $(P_1)_{\xi\perp} \ldots \underline{(P_n)}_{\xi\perp}$; here $\overline{f}(x_1, \ldots, x_n)$ is ξ or $\perp$ if some x_j, but no previous x_i, is; we write f for $\overline{f}$.

For any memory layout w and store s, we let $w \cdot s$ be the memory defined on $\mathrm{ran}(w)$ by:

$$w \cdot s(i) = s(l) \text{ for } w(l) = i$$

We extend the notation $w \cdot s$ to $s \in S_{\xi\perp}$, so that $w \cdot \xi = \xi$ and $w \cdot \perp = \perp$. A store projection is a function $\zeta\colon \mathrm{Mem}^W_{\xi\perp}$ of the form $w \mapsto w \cdot s$, for some $s \in S_{\xi\perp}$.

4.2.2 What should the denotational semantics be?

We discuss a simple example in order to explain our choice of type of the low-level denotational semantics. A straightforward semantics might have the type:

$$W \times \mathrm{Mem} \to \mathcal{H}(\mathrm{Mem}_{\xi\perp})$$

so that the meaning of a command would be a function from layouts and memories to sets of memories (modulo the use of the "error + nontermination" monad). Using our example we argue that this is unsatisfactory, and arrive at a more satisfactory alternative.

Suppose that there is a unique private location l, and that memory has four addresses, $\{1, 2, 3, 4\}$. We write s_i for the store $(l \mapsto i)$. The 4 possible layouts are $w_i = (l \mapsto i)$, for $i = 1, \ldots, 4$. Assume that d is uniform. Consider the following command:

$$c_4 \quad = \quad (1{:=}1) \ + \ (2{:=}1) \ + \ (3{:=}1) \ + \ (4{:=}1)$$

which nondeterministically guesses an address and attempts to write 1 into it. Intuitively, this command should fail to overwrite l most of the time. However, in a straightforward semantics of the above type we would have:

$$[\![c_4]\!](w_j, w_j \cdot s_0) = \{\xi, w_j \cdot s_1\} \downarrow$$

and we cannot state any quantitative property of the command, only that it sometimes fails and that it sometimes terminates.

One can rewrite the type of this semantics as:

$$\mathrm{Mem} \to \mathcal{H}(\mathrm{Mem}_{\xi\bot})^W$$

and view that as a type of functions that yield an $\mathcal{H}(\mathrm{Mem}_{\xi\bot})$-valued random variable with sample space W (the set of memory layouts) and distribution d. Thus, in this semantics, the nondeterministic choice is made after the probabilistic one —the wrong way around, as indicated in the Introduction.

It is therefore natural to reverse matters and look for a semantics of type:

$$\mathrm{Mem} \to \mathcal{H}(\mathrm{Mem}_{\xi\bot}^W)$$

now yielding a set of $\mathrm{Mem}_{\xi\bot}$-valued random variables—so, making the nondeterministic choice first. Desirable as this may be, there seems to be no good notion of composition of such functions.

Fortunately, this last problem can be overcome by changing the argument type to also be that of $\mathrm{Mem}_{\xi\bot}$-valued random variables:

$$\mathrm{Mem}_{\xi\bot}^W \to \mathcal{H}(\mathrm{Mem}_{\xi\bot}^W)$$

It turns out that with this semantics we have:

$$[\![c_4]\!](\zeta_i) = \{\zeta_\xi^1, \zeta_\xi^2, \zeta_\xi^3, \zeta_\xi^4\} \downarrow$$

where $\zeta_i(w) = w \cdot s_i$ and $\zeta_\xi^i(w) = w_i \cdot s_1$ if $w = w_i$ and $= \xi$ otherwise. We can then say that, for every nondeterministic choice, the probability of an error (or nontermination, as we are using the lower powerdomain) is 0.75.

In a further variant in the definition of the semantics, one might replace $\mathrm{Mem}_{\xi\bot}$-valued random variables by the corresponding probability distributions on $\mathrm{Mem}_{\xi\bot}$, via the natural map $\mathrm{Ind}_d \colon \mathrm{Mem}_{\xi\bot}^W \longrightarrow \mathcal{V}(\mathrm{Mem}_{\xi\bot})$ induced by the distribution d on

W. Such a semantics could have the form:

$$\mathrm{Mem} \to \mathcal{H}_{\mathcal{V}}(\mathrm{Mem}_{\xi\perp})$$

mapping memories to probability distributions on memories, where $\mathcal{H}_{\mathcal{V}}$ is a powerdomain for mixed nondeterministic and probabilistic choice as discussed above. However, such an approach would imply (incorrectly) that a new layout is chosen independently for each memory operation, rather than once and for all. In our small example with the single private location l and four addresses, it would not capture that $(1{:=}1); (2{:=}1)$ will always fail. It would treat the two assignments in $(1{:=}1); (2{:=}1)$ as two separate guesses that may both succeed. Similarly, it would treat the two assignments in $(1 := 1); (1 := 2)$ as two separate guesses where the second guess may fail to overwrite l even if the first one succeeds. With a layout chosen once and for all, on the other hand, the behavior of the second assignment is completely determined after the first assignment.

4.2.3 Denotational semantics

The denotational semantics

$$[\![e]\!] : \mathrm{Mem} \times W \to \mathbb{N}_{\xi\perp} \qquad [\![b]\!] : \mathrm{Mem} \times W \to \mathbb{B}_{\xi\perp}$$

of expressions are defined in a standard way, with, in particular, $[\![l_{\mathtt{nat}}]\!]^w_m = w(l)$, and also $[\![!e]\!]^w_m = m([\![e]\!]^w_m)$, if $[\![e]\!]^w_m \in \mathrm{dom}(m)$, and $= \xi$, otherwise, using an obvious notation for functional application. Note that these semantics never have value $\perp$.

As discussed above, the denotational semantics of commands has type:

$$[\![c]\!] {:} \mathrm{Mem}^W_{\xi\perp} \to \mathcal{H}(\mathrm{Mem}^W_{\xi\perp})$$

The definition is given in Figure 3; it makes use of two auxiliary definitions. We first define:

$$\mathrm{Ass} {:} \mathrm{Mem}_{\xi\perp} \times \mathbb{N}_{\xi\perp} \times \mathbb{N}_{\xi\perp} \to \mathbb{N}_{\xi\perp}$$

by setting $\mathrm{Ass}(m, x, y) = m[x \mapsto y]$ if $x \in \mathrm{dom}(m)$ and $= \xi$, otherwise, for $m \in \mathrm{Mem}$, $x, y \in \mathbb{N}$, and then using the function extension associated to the "error + nontermination" monad. Second, we define

$$\mathrm{Cond}(p, \theta, \theta') {:} \mathrm{Mem}^W_{\xi\perp} \to \mathcal{H}(\mathrm{Mem}^W_{\xi\perp})$$

for any $p {:} \mathrm{Mem} \times W \to \mathbb{B}_{\xi\perp}$ and $\theta, \theta' {:} \mathrm{Mem}^W_{\xi\perp} \to \mathcal{H}(\mathrm{Mem}^W_{\xi\perp})$, by:

$$\begin{aligned}\mathrm{Cond}(p, \theta, \theta')(\zeta) = \{\zeta' \mid\ & \zeta'|_{W_{\zeta,\mathrm{tt}}} \in \theta(\zeta)|_{W_{\zeta,\mathrm{tt}}}, \zeta'|_{W_{\zeta,\mathrm{ff}}} \in \theta'(\zeta)|_{W_{\zeta,\mathrm{ff}}}, \\ & \zeta'(W_{\zeta,\xi}) \subseteq \{\xi\},\ \text{and}\ \zeta'(W_{\zeta,\perp}) \subseteq \{\perp\}\}\end{aligned}$$

where $W_{\zeta,t} =_{\mathrm{def}} \{w \mid p(\zeta(w), w) = t\}$, for $t \in \mathbb{B}_{\xi\perp}$, and we apply restriction elementwise to sets of functions.

$$
\begin{array}{lll}
[\![c+c']\!](\zeta) = [\![c]\!](\zeta) \cup [\![c']\!](\zeta) & [\![c;c']\!] = [\![c']\!]^{\dagger} \circ [\![c]\!] & [\![\mathsf{skip}]\!] = \eta
\end{array}
$$

$$
\begin{array}{ll}
[\![e := e']\!](\zeta) & = \eta(\lambda w\!:\!W.\,\mathrm{Ass}(\zeta(w), [\![e]\!]^{w}_{\zeta(w)}, [\![e']\!]^{w}_{\zeta(w)})) \\
[\![\mathtt{if}\,b\,\mathtt{then}\,c\,\mathtt{else}\,c']\!] & = \mathrm{Cond}([\![b]\!], [\![c]\!], [\![c']\!]) \\
[\![\mathtt{while}\,b\,\mathtt{do}\,c]\!] & = \mu\theta\!:\!\mathrm{Mem}^{W}_{\xi\perp} \to \mathcal{H}(\mathrm{Mem}^{W}_{\xi\perp}).\mathrm{Cond}([\![b]\!], \theta^{\dagger} \circ [\![c]\!], \eta)
\end{array}
$$

Fig. 3. Low-level denotational semantics

Example 4.1 In this example, we demonstrate our low-level denotational semantics. Consider the command:

$$c_5 = l'_{\mathtt{nat}} := l_{\mathtt{nat}}; (!l'_{\mathtt{nat}}) := 1; l'_{\mathtt{nat}} := 0$$

This command stores the address of location l at location l', then reads the contents of location l' (the address of l) and writes 1 at this address, and finally resets the memory at location l' to 0. Because of this manipulation of memory locations, this command is not the direct translation of a high-level command.

Letting:

$$s_{i,j} = (l \mapsto i, l' \mapsto j) \qquad \zeta_{i,j} = w \mapsto w \cdot s_{i,j} \qquad \zeta'_i = w \mapsto w \cdot (l \mapsto i, l' \mapsto w(l))$$

we have:

$$[\![l'_{\mathtt{nat}} := l_{\mathtt{nat}}]\!](\zeta_{i,j}) = \{\zeta'_i\}\!\downarrow$$

Note that $\zeta_{i,j}$ is a store projection, but ζ'_i is not. We also have:

$$[\![(!l'_{\mathtt{nat}}) := 1]\!](\zeta'_i) = \{\zeta'_1\}\!\downarrow \qquad [\![l'_{\mathtt{nat}} := 0]\!](\zeta'_1) = \{\zeta_{1,0}\}\!\downarrow$$

In sum, we have:

$$[\![c_5]\!](\zeta_{i,j}) = \{\zeta_{1,0}\}\!\downarrow$$

□

Looking at the type of the semantics

$$[\![c]\!]\!:\!\mathrm{Mem}^{W}_{\xi\perp} \to \mathcal{H}(\mathrm{Mem}^{W}_{\xi\perp})$$

one may be concerned that there is no apparent relation between the layouts used in the input to $[\![c]\!]$ and those in its output. However, we note that the semantics could be made parametric. For every $W' \subseteq W$, replace W by W' in the definition of $[\![c]\!]$ to obtain:

$$[\![c]\!]_{W'}\!:\!\mathrm{Mem}^{W'}_{\xi\perp} \to \mathcal{H}(\mathrm{Mem}^{W'}_{\xi\perp})$$

There is then a naturality property, that the following diagram commutes for all $W'' \subseteq W' \subseteq W$:

$$\begin{tikzcd}
\mathrm{Mem}_{\xi\bot}^{W'} \arrow[r, "{[\![c]\!]_{W'}}"] \arrow[d, "\mathrm{Mem}_{\xi\bot}^{\iota}"'] & \mathcal{H}(\mathrm{Mem}_{\xi\bot}^{W'}) \arrow[d, "\mathcal{H}(\mathrm{Mem}_{\xi\bot}^{\iota})"] \\
\mathrm{Mem}_{\xi\bot}^{W''} \arrow[r, "{[\![c]\!]_{W''}}"'] & \mathcal{H}(\mathrm{Mem}_{\xi\bot}^{W''})
\end{tikzcd}$$

where $\iota : W'' \subseteq W'$ is the inclusion map. Taking $W' = W$ and W'' a singleton yields the expected relation between input and output: the value of a random variable in the output at a layout depends only on the value of the input random variable at that layout. The naturality property suggests re-working the low level denotational semantics in the category of presheaves over sets of layouts, and this may prove illuminating (see [12] for relevant background).

4.3 Operational semantics

As a counterpart to the denotational semantics, we give a deterministic operational semantics using oracles to make choices. The oracles are elements of the set Ω of infinite lists of tokens L (for "left") and R (for "right"). A low-level state σ is:

- a triple $\langle c, m, \pi\rangle$ of a command c, a memory m, and an oracle π; or
- a pair $\langle m, \pi\rangle$ of a memory m and an oracle π; or
- the error element ξ.

Transitions are given relative to a layout, so we write:

$$w \models \sigma \rightarrow \sigma'$$

The rules are given in Figure 4. This semantics is deterministic for each choice of layout. We write $w \models \sigma \Rightarrow \sigma'$ for the transitive closure of the transition relation (for a given layout).

Example 4.2 Consider the command c_4 introduced in Section 4.2.2, with added parentheses for disambiguation:

$$c_4 \quad = \quad (1{:=}1) \ + \ ((2{:=}1) \ + \ ((3{:=}1) \ + \ ((4{:=}1))))$$

We have:

$$\begin{array}{llll}
w_1 \models \langle c_4, w_1 \cdot s_k, L\pi\rangle & \rightarrow \langle w_1 \cdot s_1, \pi\rangle & w_j \models \langle c_4, w_j \cdot s_k, L\pi\rangle & \rightarrow \xi \ (j \neq 1) \\
w_2 \models \langle c_4, w_2 \cdot s_k, RL\pi\rangle & \Rightarrow \langle w_2 \cdot s_1, \pi\rangle & w_j \models \langle c_4, w_j \cdot s_k, RL\pi\rangle & \Rightarrow \xi \ (j \neq 2) \\
w_3 \models \langle c_4, w_3 \cdot s_k, RRL\pi\rangle & \Rightarrow \langle w_3 \cdot s_1, \pi\rangle & w_j \models \langle c_4, w_j \cdot s_k, RRL\pi\rangle & \Rightarrow \xi \ (j \neq 3) \\
w_4 \models \langle c_4, w_4 \cdot s_k, RRR\pi\rangle & \Rightarrow \langle w_4 \cdot s_1, \pi\rangle & w_j \models \langle c_4, w_j \cdot s_k, RRR\pi\rangle & \Rightarrow \xi \ (j \neq 4)
\end{array}$$

□

$$\frac{[\![e]\!]^w_m \in \mathrm{dom}(m) \text{ and } [\![e']\!]^w_m \neq \xi}{w \models \langle e := e', m, \pi\rangle \to \langle m[[\![e]\!]^w_m \mapsto [\![e']\!]^w_m], \pi\rangle} \qquad \frac{[\![e]\!]^w_m \notin \mathrm{dom}(m) \text{ or } [\![e']\!]^w_m = \xi}{w \models \langle e := e', m, \pi\rangle \to \xi}$$

$$\frac{[\![b]\!]^w_m = \mathtt{tt}}{w \models \langle \mathtt{if}\, b\, \mathtt{then}\, c\, \mathtt{else}\, c', m, \pi\rangle \to \langle c, m, \pi\rangle}$$

$$\frac{[\![b]\!]^w_m = \mathtt{ff}}{w \models \langle \mathtt{if}\, b\, \mathtt{then}\, c\, \mathtt{else}\, c', m, \pi\rangle \to \langle c', m, \pi\rangle} \qquad \frac{[\![b]\!]^w_m = \xi}{w \models \langle \mathtt{if}\, b\, \mathtt{then}\, c\, \mathtt{else}\, c', m, \pi\rangle \to \xi}$$

$$w \models \langle \mathsf{skip}, m, \pi\rangle \to \langle m, \pi\rangle \qquad \frac{w \models \langle c, m, \pi\rangle \to \langle c', m', \pi'\rangle}{w \models \langle c; c'', m, \pi\rangle \to \langle c'; c'', m', \pi'\rangle}$$

$$\frac{w \models \langle c, m, \pi\rangle \to \langle m', \pi'\rangle}{w \models \langle c; c'', m, \pi\rangle \to \langle c'', m', \pi'\rangle} \qquad \frac{w \models \langle c, m, \pi\rangle \to \xi}{w \models \langle c; c'' m, \pi\rangle \to \xi}$$

$$w \models \langle c + c', m, L\pi\rangle \to \langle c, m, \pi\rangle \qquad w \models \langle c + c', m, R\pi\rangle \to \langle c', m, \pi\rangle$$

$$\frac{[\![b]\!]^w_m = \mathtt{ff}}{w \models \langle \mathtt{while}\, b\, \mathtt{do}\, c, m, \pi\rangle \to \langle m, \pi\rangle}$$

$$\frac{[\![b]\!]^w_m = \mathtt{tt}}{w \models \langle \mathtt{while}\, b\, \mathtt{do}\, c, m, \pi\rangle \to \langle c; \mathtt{while}\, b\, \mathtt{do}\, c, m, \pi\rangle} \qquad \frac{[\![b]\!]^w_m = \xi}{w \models \langle \mathtt{while}\, b\, \mathtt{do}\, c, m, \pi\rangle \to \xi}$$

Fig. 4. Low-level operational semantics

Using the operational semantics, we can define an evaluation function:

$$\mathrm{Eval} : \mathrm{Com} \times W \times \mathrm{Mem} \times \Omega \to \mathrm{Mem}_{\xi\perp}$$

by:

$$\mathrm{Eval}(c, w, m, \pi) = \begin{cases} m' & (w \models \langle c, m, \pi\rangle \Rightarrow \langle m', \pi'\rangle) \\ \xi & (w \models \langle c, m, \pi\rangle \Rightarrow \xi) \\ \perp & (\text{otherwise}) \end{cases}$$

We then define

$$\mathrm{Eval}_{\mathrm{ran}} : \mathrm{Com} \times \mathrm{Mem}^W_{\xi\perp} \to \Omega \to \mathrm{Mem}^W_{\xi\perp}$$

by:

$$\mathrm{Eval}_{\mathrm{ran}}(c, \zeta)(\pi)(w) = \begin{cases} \mathrm{Eval}(w, c, \zeta(w), \pi) & (\zeta(w) \in \mathrm{Mem}) \\ \zeta(w) & (\text{otherwise}) \end{cases}$$

Making use of the image functional $\mathrm{Im}_X : X^\Omega \to \mathcal{P}(X)$, where $\mathrm{Im}_X(f) = f(\Omega)$, we can state the consistency of the operational and denotational semantics:

Proposition 4.3 (Operational/denotational consistency) *For c a command*

and ζ a function in $\mathrm{Mem}^W_{\xi\perp}$*, we have:*

$$[\![c]\!](\zeta) = \mathrm{Im}_{\mathrm{Mem}^W_{\xi\perp}}(\mathrm{Eval}_{\mathrm{ran}}(c, \zeta)) \downarrow$$

The evaluation function yields operational correlates of the other possible denotational semantics discussed in Section 4.2.2, similarly, using image or induced distribution functionals. For example, for the first of those semantics, by currying Eval and composing, one obtains:

$$\begin{tikzcd}\mathrm{Com} \times W \times \mathrm{Mem} \arrow[r, "\mathrm{curry}(\mathrm{Eval})"] & \mathrm{Mem}^{\Omega}_{\xi\perp} \arrow[r, "\mathrm{Im}_{\mathrm{Mem}_{\xi\perp}}"] & \mathcal{P}(\mathrm{Mem}_{\xi\perp})\end{tikzcd}$$

Using such operational correlates, one can verify operational versions of the assertions made in Section 4.2.2 about the inadequacies of those semantics.

4.4 Implementation relations and equivalences

Much as in the high-level language, we define a contextual implementation relation and a simulation relation for the low-level language. The low-level definitions refer to layouts, and in some cases include conditions on induced probabilities.

4.4.1 Contextual pre-order

Again, the contextual pre-order $c \sqsubseteq_L c'$ may be interpreted as saying that c "refines" (or "implements") c', in the sense that the publicly observable outcomes that c can produce are a subset of those that c' permits, in every public context. In comparison with definition for the high-level language, however, c and c' are not applied to an arbitrary initial store but rather to a function from layouts to memories (extended with "error + nontermination"), and they produce sets of such functions. We restrict attention to argument functions induced by stores, in the sense that they are store projections of the form $w \mapsto w \cdot s$. Thus, let $f = [\![C[c]]\!]$ and $f' = [\![C[c']]\!]$ for an arbitrary public context C, and let s be a store; then (roughly) for every ζ in $f(w \mapsto w \cdot s)$ there exists ζ' in $f'(w \mapsto w \cdot s)$ such that, for any w, $\zeta(w)$ and $\zeta'(w)$ coincide on public locations.

The treatment of error and nontermination introduces a further complication. Specifically, we allow that ζ produces an error or diverges with sufficient probability ($\geq \delta$), and that ζ' produces an error with sufficient probability ($\geq \delta$), as an alternative to coinciding on public locations.

Therefore, we define $\sqsubseteq_L$ and some auxiliary notation and relations:

- Let $m \in \mathrm{Mem}$ be a memory. We write m_L for the restriction of m to $\mathrm{ran}(w_p)$, extending the notation to $\mathrm{Mem}_{\xi\perp}$ as usual.
- Suppose $\zeta \in \mathrm{Mem}^W_{\xi\perp}$; we write ζ_L for the partial function defined by $\zeta_L(w) = \zeta(w)_L$.
- For $X, Y \in \mathcal{H}(\mathrm{Mem}^W_{\xi\perp})$, we write that $X \leq_L Y$ when, for every $\zeta \in X$, there exists $\zeta' \in Y$ such that:
  · $\zeta_L \leq \zeta'_L$, or
  · $P(\zeta(w) \in \{\xi, \perp\}) \geq \delta$ and $P(\zeta'(w) = \xi) \geq \delta$.

- For $f, f' \in \mathrm{Mem}^W_{\xi\perp} \to \mathcal{H}(\mathrm{Mem}^W_{\xi\perp})$, we write $f \leq_L f'$ when, for all $s \in S$, we have:

$$f(w \mapsto w{\cdot}s) \leq_L f'(w \mapsto w{\cdot}s)$$

- Finally, we write $c \sqsubseteq_L c'$ when, for every public command context C, $[\![C[c]]\!] \leq_L [\![C[c']]\!]$.

4.4.2 Simulation

As in the high-level language, we introduce a simulation relation $\preceq$. This relation works only on commands whose outcomes on inputs that are store projections are themselves store projections; nevertheless, simulation remains a useful tool for proofs.

We define $\varpi : S_{\xi\perp} \to \mathcal{H}(\mathrm{Mem}^W_{\xi\perp})$ by:

$$\begin{aligned} \varpi(\perp) &= \{w \mapsto \perp\}\downarrow \\ \varpi(s) &= \{w \mapsto w{\cdot}s\}\downarrow \\ \varpi(\xi) &= \{\zeta | P(\zeta(w) = \xi) \geq \delta\}\downarrow \end{aligned}$$

For every $X \in \mathcal{H}(\mathrm{Mem}^W_{\xi\perp})$, we say that X is a store projection set when there exists $Y \in \mathcal{H}(S_{\xi\perp})$ such that

$$\varpi(Y \setminus \{\xi\})\downarrow \subseteq X \subseteq \varpi(Y)\downarrow$$

and

$$\xi \in Y \Rightarrow \exists \zeta \in X.\, P(\zeta(w) = \xi) \geq \delta$$

In that case, we write $\chi(X) = Y$ for the unique such Y (we have $s \in Y$ if, and only if, $w \mapsto w \cdot s \in X$ and $\xi \in Y$ if, and only if, $\exists \zeta \in X, P(\zeta(w) = \xi) \geq \delta$).

Much as in the high-level language, when R is a relation over $\mathcal{H}(S_{H\perp})$, we have that R^+ is a relation over $\mathcal{H}(S_\perp)$, and we extend it to a relation over $\mathcal{H}(S_{\xi\perp})$ as follows. For any $X, Y \in \mathcal{H}(S_{\xi\perp})$:

$$XR^+Y \equiv_{\mathrm{def}} (\xi \in X \Rightarrow \xi \in Y) \wedge ((X \setminus \xi)R^+(Y \setminus \xi))$$

Then we define a relation $R^\times$ over $\mathcal{H}(\mathrm{Mem}^W_{\xi\perp})$ as follows: for any $X, Y \in \mathcal{H}(\mathrm{Mem}^W_{\xi\perp})$, $XR^\times Y$ holds if, and only if, X and Y are store projection sets and $\chi(X)R^+\chi(Y)$ holds. If R is closed under increasing ω-sups (respectively, is right-closed under $\leq$ and closed under binary unions) the same holds for $R^\times$ (with $\leq$ restricted to store projection sets). If R is reflexive, then $R^\times$ is reflexive on store projection sets. We also have:

Fact 4.4 *For all $X, Y \in \mathcal{H}(\mathrm{Mem}^W_{\xi\perp})$, if $XR^\times Y$ then $X \leq_L Y$.*

For any $f, f' : \mathrm{Mem}^W_{\xi\perp} \to \mathcal{H}(\mathrm{Mem}^W_{\xi\perp})$ and relation R over $\mathcal{H}(S_{H\perp})$ we write that $f \preceq_R f'$ when:

$$\forall X, Y \in \mathcal{H}(\mathrm{Mem}^W_{\xi\perp}).\, XR^\times Y \Rightarrow f^\dagger(X)R^\times f'^\dagger(Y)$$

Finally, we write that $f \preceq f'$ if $f \preceq_R f'$ for some reflexive R closed under increasing ω-sups, right-closed under $\leq$, and closed under binary unions.

4.4.3 Contextual pre-order vs. simulation

The contextual pre-order coincides with the simulation relation, but only for commands whose semantics sends store projections to store projection sets. Formally, we say that a given function $f : \mathrm{Mem}^W_{\xi\perp} \to \mathcal{H}(\mathrm{Mem}^W_{\xi\perp})$ preserves store projections if, for every $s \in S$, $f(w \mapsto w \cdot s)$ is a store projection set. The coincidence remains quite useful despite this restriction, which in particular is not an impediment to our overall goal of relating the low-level language to the high-level language.

Theorem 4.5 *Let c and c' be two commands of the low-level language such that $[\![c]\!]$ and $[\![c']\!]$ preserve store projections. Then $c \sqsubseteq_L c'$ if and only if $[\![c]\!] \preceq [\![c']\!]$.*

Example 4.6 Suppose that there is only one private location, and consider the two commands:

$$c_4 = (1 := 1) \ + \ (2 := 1) \ + \ (3 := 1) \ + \ (4 := 1) \qquad c_6 = (1 := 1); (2 := 1)$$

As seen above, we have $[\![c_4]\!](\zeta_i) = \{\zeta^1_\xi, \zeta^2_\xi, \zeta^3_\xi, \zeta^4_\xi\}\downarrow$. We also have $[\![c_6]\!](\zeta_i) = \{w \mapsto \xi\}\downarrow$. Since $P(\zeta^i_\xi(w) = \xi) \geq \delta$, we can verify that c_4 and c_6 are equivalent. (Thus, a nondeterministic guess is no better than failure.) □

5 High and low

In this section we investigate the relation between the high-level language and the low-level language. Specifically, we define a simple translation from the high-level language to the low-level language, then we study its properties.

We define the compilation of high-level commands c (expressions e, boolean expressions b) to low-level commands $c^\downarrow$ (expressions $e^\downarrow$ and boolean expressions $b^\downarrow$) by setting: $(!l_{\mathtt{loc}})^\downarrow = !l_{\mathtt{nat}}$, $(l_{\mathtt{loc}} := e)^\downarrow = l_{\mathtt{nat}} := e^\downarrow$, and proceeding homomorphically in all other cases (e.g., $(e + e')^\downarrow = e^\downarrow + e'^\downarrow$). Crucially, this compilation function, which is otherwise trivial, transforms high-level memory access to low-level memory access.

Lemma 5.1 *Let c be a high-level command. Then $[\![c^\downarrow]\!]$ preserves store projections.*

Theorem 5.2 relates the simulation relations of the two languages. It states that a high-level command c simulates another high-level command c, with respect to all public contexts of the high-level language, if and only if the compilation of c simulates the compilation of c', with respect to all public contexts of the low-level language.

Theorem 5.2 *Let c and c' be two high-level commands. Then $[\![c]\!] \preceq [\![c']\!]$ if and only if $[\![c^\downarrow]\!] \preceq [\![c'^\downarrow]\!]$.*

Our main theorem, Theorem 5.3, follows from Theorem 5.2, the two previous theorems, and the lemma. Theorem 5.3 is analogous to Theorem 5.2, but refers to the contextual pre-orders: a high-level command c implements another high-level command c', with respect to all public contexts of the high-level language, if and only if the compilation of c implements the compilation of c', with respect to all public contexts of the low-level language.

Theorem 5.3 (Main theorem) *Let c and c' be two high-level commands. Then $c \sqsubseteq_L c'$ if and only if $c^{\downarrow} \sqsubseteq_L c'^{\downarrow}$.*

Theorem 5.3 follows from Theorem 5.2, the two previous theorems, and the lemma. The low-level statement is defined in terms of the probability δ that depends on the distribution on memory layouts. When δ is close to 1, the statement indicates that, from the point of view of a public context (that is, an attacker), the compilation of c behaves like an implementation of the compilation of c'. This implementation relation holds despite the fact that the public context may access memory via natural-number addresses, and thereby (with some probability) read or write private data of the commands. The public context may behave adaptively, with memory access patterns chosen dynamically, for instance attempting to exploit correlations in the distribution of memory layouts. The public context may also give "unexpected" values to memory addresses, as in practical attacks; the theorem implies that such behavior is no worse at the low level than at the high level.

For example, for the commands c_0 and c_1 of Example 3.1, the theorem enables us to compare how their respective compilations behave, in an arbitrary public low-level context. Assuming that δ is close to 1, the theorem basically implies that a low-level attacker that may access memory via natural-number addresses cannot distinguish those compilations. Fundamentally, this property holds simply because the attacker can read or write the location h only with low probability.

6 Conclusion

A few recent papers investigate the formal properties of layout randomization, like ours [19,3,13,2]. They do not consider nondeterministic choice, and tend to reason operationally. However, the work of Jagadeesan et al. includes some semantic elements that partly encouraged our research; specifically, that work employs trace equivalence as a proof technique for contextual equivalence.

In this paper we develop a semantic approach to the study of layout randomization. Our work concerns nondeterministic languages, for which this approach has proved valuable in reconciling probabilistic choice with nondeterministic choice. However, the approach is potentially more general. In particular, the study of concurrency with nondeterministic scheduling would be an attractive next step. Also, extending our work to higher-order computation presents an interesting challenge.

References

[1] M. Abadi and L. Lamport. The existence of refinement mappings. *TCS*, 82(2):253–284, 1991.

[2] M. Abadi and J. Planul. On layout randomization for arrays and functions. In *POST*, volume 7796 of *LNCS*, pages 167–185. Springer, 2013.

[3] M. Abadi and G. D. Plotkin. On protection by layout randomization. *ACM Transactions on Information and System Security*, 15(2):8:1–8:29, 2012.

[4] R. Canetti et al. Analyzing security protocols using time-bounded task-pioas. *Discrete Event Dynamic Systems*, 18(1):111–159, 2008.

[5] W. P. de Roever and K. Engelhardt. *Data Refinement: Model-oriented Proof Theories and their Comparison*, volume 46 of *Cambridge Tracts in Theo. Comp. Sci.* CUP, 1998.

[6] P. Druschel and L. L. Peterson. High-performance cross-domain data transfer. Technical Report TR 92-11, Department of Computer Science, The University of Arizona, 1992.

[7] Ú. Erlingsson. Low-level software security: Attacks and defenses. In *FOSAD IV Tutorial Lectures*, volume 4677 of *LNCS*, pages 92–134. Springer, 2007.

[8] S. Forrest et al. Building diverse computer systems. In *6th Workshop on Hot Topics in Operating Systems*, pages 67–72, 1997.

[9] G. Gierz et al. *Continuous lattices and domains*, volume 93 of *Encyclopaedia of mathematics and its applications.* CUP, 2003.

[10] S. Goldwasser and S. Micali. Probabilistic encryption. *JCSS*, 28:270–299, 1984.

[11] J. Goubault-Larrecq. Prevision domains and convex powercones. In *FoSSaCS*, volume 4962 of *LNCS*, pages 318–333. Springer, 2008.

[12] M. Jackson. *A sheaf theoretic approach to measure theory.* PhD thesis, U. Pitt., 2006.

[13] R. Jagadeesan et al. Local memory via layout randomization. In *Proc. of the 24th CSFS*, pages 161–174, 2011.

[14] N. Klarlund and F. B. Schneider. Proving nondeterministically specified safety properties using progress measures. *Information and Computation*, 107(1):151–170, 1993.

[15] P. Lincoln et al. A probabilistic poly-time framework for protocol analysis. In *Proceedings of the Fifth ACM Conference on Computer and Communications Security*, pages 112–121, 1998.

[16] M. W. Mislove. On combining probability and nondeterminism. *ENTCS*, 162:261–265, 2006.

[17] J. C. Mitchell et al. A probabilistic polynomial-time process calculus for the analysis of cryptographic protocols. *TCS*, 353(1-3):118–164, 2006.

[18] PaX Project. The PaX project, 2004. http://pax.grsecurity.net/.

[19] R. Pucella and F. B. Schneider. Independence from obfuscation: A semantic framework for diversity. *Journal of Computer Security*, 18(5):701–749, 2010.

[20] A. Sabelfeld and D. Sands. Probabilistic noninterference for multi-threaded programs. In *CSFW*, pages 200–214, 2000.

[21] R. Tix et al. Semantic domains for combining probability and non-determinism. *ENTCS*, 222:3–99, 2009.

Normalization by Evaluation and Algebraic Effects

Danel Ahman[1]

Laboratory for Foundations of Computer Science
University of Edinburgh

Sam Staton[2]

Computer Laboratory
University of Cambridge

Abstract

We examine the interplay between computational effects and higher types. We do this by presenting a normalization by evaluation algorithm for a language with function types as well as computational effects. We use algebraic theories to treat the computational effects in the normalization algorithm in a modular way. Our algorithm is presented in terms of an interpretation in a category of presheaves equipped with partial equivalence relations. The normalization algorithm and its correctness proofs are formalized in dependent type theory (Agda).

Keywords: Algebraic effects, Type theory, Normalization by evaluation, Presheaves, Monads

1 Introduction

When studying computer programs it is often appropriate to consider them up-to some equations. In this paper we consider an equational theory for impure functional programs. By finding a class of normal forms for this equational theory, we are able to understand and manipulate the notions under study directly. Moreover, it has been proposed that normalization algorithms are of use in partial evaluation: if a program fragment with free variables is normalized at compile-time then it will typically run faster.

[1] Email: d.ahman@ed.ac.uk

[2] Email: sam.staton@cl.cam.ac.uk

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.com/locate/entcs

To be more precise, we introduce a small program in an ML-like language.

```
(fn (g:(unit -> unit) -> unit)
   => g (if recv()=0 then fn x => send 0 ; h x else fn y => send 1))
(fn (f:unit -> unit) => f () ; f ())                                    (*)
```

Here `recv:unit->bit` and `send:bit->unit` are network communication primitives, as in Concurrent ML [39], and `h:unit->unit` is a free identifier of function type. Notice that we cannot naively compile and run this program to find out what it does, because it has a free identifier `h`, and because its execution will depend on what is received from the network.

Before we normalize the program, we translate it to an intermediate language which makes the evaluation order clear. We also remove the `bit` type from the program, since it complicates the normalization process and is orthogonal to what we are investigating. (We return to the issue of sum types in §6). We eliminate the need for a `bit` type by using algebraic operations, following [36]: we replace `(if recv()=0 then` M `else` N`)` by $\mathsf{inp}[M, N]$, replace `(send 0 ;` M`)` by $\mathsf{out}_0[M]$ and `(send 1 ;` M`)` by $\mathsf{out}_1[M]$. Thus the program (*) becomes

$$\begin{aligned}&\Big(\mathtt{fn}\, g{:}((\mathtt{unit} \rightharpoonup \mathtt{unit}) \rightharpoonup \mathtt{unit}) \Rightarrow \mathsf{inp}[\mathtt{return}\,\mathtt{fn}\,x \Rightarrow \mathsf{out}_0[h\,x], \\ &\qquad\qquad\qquad\qquad \mathtt{return}\,\mathtt{fn}\,y \Rightarrow \mathsf{out}_1[\mathtt{return}\,\langle\rangle]]\ \ \mathtt{to}\, f.\, g\, f\Big) \qquad (\dagger)\\ &\Big(\mathtt{fn}\, f{:}(\mathtt{unit} \rightharpoonup \mathtt{unit}) \Rightarrow f\,\langle\rangle\ \mathtt{to}\, y.\, f\,\langle\rangle\Big)\end{aligned}$$

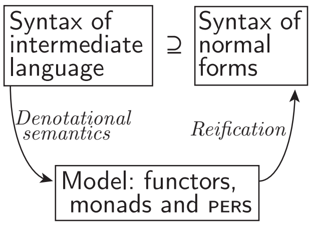
The intermediate language (§2) has a straightforward equational theory, including β and η equality. The program (†) is not in normal form for these equations, e.g. it has a β-redex. Our normalization algorithm yields the following program:

$$\mathsf{inp}[\mathsf{out}_0[h\,\langle\rangle\ \mathtt{to}\, x.\, \mathsf{out}_0[h\,\langle\rangle\ \mathtt{to}\, y.\, \mathtt{return}\,\langle\rangle]], \mathsf{out}_1[\mathsf{out}_1[\mathtt{return}\,\langle\rangle]]] \qquad (\ddagger)$$

So we discover what the program (*) does: it inputs a bit from the network. If that bit is 0 then it outputs 0, calls h, outputs 0, and calls h again. If the bit from the network is 1 then it outputs 1 twice.

Notice how we are describing computational effects with an algebraic signature: inp is a binary operation, and out_0, out_1 are unary operations. A crucial observation is that the same normalization algorithm works if we begin with a different algebraic signature of computational effects. Many other effects have been described in an algebraic way, including non-determinism, probability, memory access [36,35,26] and logic programming [41]. Our framework is a general one for all these examples.

1.1 The essence of normalization by evaluation


Syntax of intermediate language
⊇
Syntax of normal forms
Denotational semantics
Reification
Model: functors, monads and PERS

We define our normalization algorithm in §3 using the paradigm of normalization by evaluation (NBE). The ideas of NBE were first discussed by Martin-Löf [25] and later developed by Berger and Schwichtenberg [8]. There are two key ingredients: (1) a denotational semantics of the programming language in an executable type theory (Agda[3]) in which terms are automatically normalized; (2) a “reification” function which takes inhabitants of the denotational semantics back to terms of intermediate language in a sub-grammar of normal forms.

1.2 Components of denotational semantics

There are three important components to our denotational semantics for NBE:

1. Semantics in a functor category: We follow the general paradigm of structuring denotational semantics by finding a category and interpreting types as objects and programs as morphisms between objects. Following [14,3,9], we base our denotational semantics on the category $\mathsf{Set}^{\mathsf{Ren}}$ of functors from a category Ren of contexts and renamings between them, to the category of sets. This category behaves very much like the category of sets, but has extra features that allow us to take care over interpreting terms with free identifiers. The key feature of $\mathsf{Set}^{\mathsf{Ren}}$ is that there is a distinguished object $\mathsf{Ren}(\tau, -)$ for each type τ of the intermediate language, and this object behaves like a special set of identifiers of type τ.

2. A residualizing monad: Our intermediate language is a variation on Moggi’s monadic metalanguage, and we structure our denotational semantics using a monad. Following Plotkin and Power [35], we build the monad from operations in the algebraic signature describing the computational effects. However, for NBE we must add more into our monad: following Filinski’s pioneering work [13] and subsequent developments [21,5], we also incorporate the effects of applying an identifier of function type to an argument. For instance, in the normal form (‡) above, although the result of the call to h is ignored, the function call may produce side effects, depending on what h stands for. We thus keep the ‘residual’ function call, which cannot be normalized any further.

3. Using PERs to account for equations on effect terms: In addition to operations in algebraic signatures, many computational effects are described with additional equations specifying their computational behaviour. Following [9,33], we accommodate such effects in our NBE algorithm by considering presheaves whose codomains are equipped with partial equivalence relations (PERs). This is a particularly elegant approach because from the perspective of the NBE algorithm, we can naively work with sets, and then refer to the PERs when justifying the correctness of the algorithm.

1.3 Contributions

Our main contribution is to build a normalization algorithm for our effectful functional language out of this semantic analysis. The three components of our denotational semantics (§1.2) have not been combined before. By combining (1) and (2) we achieve a clean and modular mathematical account of Filinski’s ideas of residuation in monads. By combining (2) and (3) we are able to analyze equations and normalization at the level of effects (§5), separately from equations and normalization of the functional aspects of the language.

[3] Agda implementation of our NBE: https://github.com/danelahman/Normalization-By-Evaluation

We also present a proof of correctness of the normalization algorithm. Our proof uses logical relations, and further exploits the tight connection between the residualizing monad and the syntax of normal forms.

Acknowledgments: We are especially grateful to James Chapman for suggestions on the Agda formalization. We also thank Sam Lindley and Andy Pitts for useful discussions. We also thank Pierre Clairambault who pointed us to Okada and Scott's undecidability result. The core of this work appeared in the first author's MPhil dissertation.

First author's participation at the conference was supported by Estonian national scholarship program Kristjan Jaak, which is funded and managed by Archimedes Foundation in collaboration with the Ministry of Education and Research.

2 A programming language with algebraic effects

We introduce a syntax and equational theory for a higher-order programming language which incorporates computational effects using algebraic theories, following [35]. Our language is based on the call-by-value paradigm. The evaluation order is totally explicit, so it is more of an intermediate language than a front-end. The language is based on Moggi's monadic metalanguage [29], following the analysis by Levy, Power and Thielecke [20] (see also [17,19,28,38]).

2.1 Algebraic effects

We describe simple effects involved in computation using algebraic signatures [36]. For example, we can describe the effects involved in input/output of bits over a fixed communication channel with a binary operation inp and unary operations out_0, out_1. The algebraic expression $\mathsf{inp}[M, N]$ describes a computation that first reads a bit from the channel and then proceeds as the computation M if it is 0, or as N if it is 1. The expression $\mathsf{out}_0[M]$ describes a computation that outputs a bit 0 to the channel and then proceeds as M.

For another example, we can describe the effects of non-determinism with a binary operation $\oplus$, with the understanding that $M \oplus N$ describes a computation that behaves either as M or as N.

Formally, an *algebraic signature* consists of a set Op of operations together with an assignment of arities $\mathrm{ar} : \mathsf{Op} \to \mathbb{N}$. For input/output, let $\mathsf{Op} \overset{\text{def}}{=} \{\mathsf{inp}, \mathsf{out}_0, \mathsf{out}_1\}$ and $\mathrm{ar}(\mathsf{inp}) \overset{\text{def}}{=} 2$, $\mathrm{ar}(\mathsf{out}_0) \overset{\text{def}}{=} 1$, $\mathrm{ar}(\mathsf{out}_1) \overset{\text{def}}{=} 1$. For non-determinism, let $\mathsf{Op} \overset{\text{def}}{=} \{\oplus\}$ and $\mathrm{ar}(\oplus) \overset{\text{def}}{=} 2$.

One would typically impose equations, such as idempotency, commutativity and associativity of $\oplus$. We postpone a discussion on this until §5. In §6 we discuss more general kinds of algebraic theories involving value parameters and variable binding.

2.2 Extending algebraic effects to a call-by-value language with higher types

The algebraic analysis of effects involves a class of computations of unspecified type. We now describe a typed language, for time being with product and function types:

$$\sigma, \tau \in \mathsf{Ty} \ ::= \ \mathtt{unit} \mid \sigma * \tau \mid \sigma \rightharpoonup \tau \ .$$

We use a harpoon symbol for the function type $\sigma \rightharpoonup \tau$ to emphasise that a function may have side effects. (Moggi's [29] notation for this is $\sigma \to T(\tau)$. Conversely in our language the thunking construction $(\mathtt{unit} \rightharpoonup (-))$ is a monad.)

We have not included other types, such as sums or recursive types, because our main aim in this paper is to present a clear underlying framework for NBE for effectful languages with algebraic effects. We return to this in §6.

A typing context is a list of types annotated with variable names x, y, z. We have no need to consider untyped terms, so we immediately provide a rule-based definition of typed terms in context. Following [20], there are two typing judgements: one for values $\Gamma \vDash_{\mathsf{v}} V : \tau$ and one for producers $\Gamma \vDash_{\mathsf{p}} M : \tau$. The idea is that a value is something that has no effects, whereas a producer may have side effects.

$$\frac{}{\Gamma, x : \tau, \Gamma' \vDash_{\mathsf{v}} x : \tau} \qquad \frac{\Gamma \vDash_{\mathsf{v}} V_1 : \tau_1 \quad \Gamma \vDash_{\mathsf{v}} V_2 : \tau_2}{\Gamma \vDash_{\mathsf{v}} \langle V_1, V_2 \rangle : \tau_1 * \tau_2} \qquad \frac{\Gamma, x : \sigma \vDash_{\mathsf{p}} N : \tau}{\Gamma \vDash_{\mathsf{v}} \mathtt{fn}\, x{:}\sigma \Rightarrow N : \sigma \rightharpoonup \tau}$$

$$\frac{}{\Gamma \vDash_{\mathsf{v}} \langle\rangle : \mathtt{unit}} \qquad \frac{\Gamma \vDash_{\mathsf{v}} V : \tau_1 * \tau_2}{\Gamma \vDash_{\mathsf{v}} \mathtt{\#}_i\, V : \tau_i} \qquad \frac{\Gamma \vDash_{\mathsf{v}} V : \sigma \rightharpoonup \tau \quad \Gamma \vDash_{\mathsf{v}} W : \sigma}{\Gamma \vDash_{\mathsf{p}} V\, W : \tau}$$

$$\frac{\Gamma \vDash_{\mathsf{v}} V : \tau}{\Gamma \vDash_{\mathsf{p}} \mathtt{return}\, V : \tau} \qquad \frac{\Gamma \vDash_{\mathsf{p}} M : \sigma \quad \Gamma, x : \sigma \vDash_{\mathsf{p}} N : \tau}{\Gamma \vDash_{\mathsf{p}} M\, \mathtt{to}\, x.\, N : \tau} \qquad \frac{\Gamma \vDash_{\mathsf{p}} M_1 : \tau \ \ \ldots \ \ \Gamma \vDash_{\mathsf{p}} M_n : \tau}{\Gamma \vDash_{\mathsf{p}} \mathsf{op}_\tau[M_1, \ldots, M_n] : \tau}$$

There is an instance of the bottom-right rule for each n-ary operation $\mathsf{op} \in \mathsf{Op}$ and each type τ. For instance, with the input/output signature we have this syntax:

$$\frac{\Gamma \vDash_{\mathsf{p}} M : \tau \quad \Gamma \vDash_{\mathsf{p}} M : \tau}{\Gamma \vDash_{\mathsf{p}} \mathsf{inp}[M, N] : \tau} \qquad \frac{\Gamma \vDash_{\mathsf{p}} M : \tau}{\Gamma \vDash_{\mathsf{p}} \mathsf{out}_0[M] : \tau} \qquad \frac{\Gamma \vDash_{\mathsf{p}} M : \tau}{\Gamma \vDash_{\mathsf{p}} \mathsf{out}_1[M] : \tau}$$

2.3 Equational theory

The equational theory of this language is built from the $\beta\eta$-equations of the λ-calculus, the laws of Kleisli composition (e.g. [20,29]), and algebraicity [38, §3.3]. We have elided the usual laws of reflexivity, symmetry, transitivity, and congruence.

$$\frac{\Gamma \vDash_{\mathsf{v}} V_1 : \tau_1 \quad \Gamma \vDash_{\mathsf{v}} V_2 : \tau_2}{\Gamma \vDash_{\mathsf{v}} \mathtt{\#}_i \, \langle V_1, V_2 \rangle \equiv V_i : \tau_i} \qquad \frac{\Gamma \vDash_{\mathsf{v}} V : \tau_1 * \tau_2}{\Gamma \vDash_{\mathsf{v}} V \equiv \langle \mathtt{\#}_1 \, V, \mathtt{\#}_2 \, V \rangle : \tau_1 * \tau_2} \qquad \frac{\Gamma \vDash_{\mathsf{v}} V : \mathtt{unit}}{\Gamma \vDash_{\mathsf{v}} V \equiv \langle\rangle : \mathtt{unit}}$$

$$\frac{\Gamma, x : \sigma \vDash_{\mathsf{p}} M : \tau \quad \Gamma \vDash_{\mathsf{v}} V : \sigma}{\Gamma \vDash_{\mathsf{p}} (\mathtt{fn}\, x{:}\sigma \Rightarrow M)\, V \equiv M[V/x] : \tau} \qquad \frac{\Gamma \vDash_{\mathsf{v}} V : \sigma \rightharpoonup \tau}{\Gamma \vDash_{\mathsf{v}} V \equiv \mathtt{fn}\, x{:}\sigma \Rightarrow (V\, x) : \sigma \rightharpoonup \tau}$$

$$\frac{\Gamma \vDash_{\mathsf{v}} V : \sigma \quad \Gamma, x : \sigma \vDash_{\mathsf{p}} N : \tau}{\Gamma \vDash_{\mathsf{p}} \mathtt{return}\, V \,\mathtt{to}\, x.\, N \equiv N[V/x] : \tau} \qquad \frac{\Gamma \vDash_{\mathsf{p}} M : \tau}{\Gamma \vDash_{\mathsf{p}} M \equiv M \,\mathtt{to}\, x.\, \mathtt{return}\, x : \tau}$$

$$\frac{\Gamma \vDash_{\mathsf{p}} M : \sigma \quad \Gamma, x : \sigma \vDash_{\mathsf{p}} N : \tau \quad \Gamma, y : \tau \vDash_{\mathsf{p}} P : \rho}{\Gamma \vDash_{\mathsf{p}} (M \,\mathtt{to}\, x.\, N) \,\mathtt{to}\, y.\, P \equiv M \,\mathtt{to}\, x.\, (N \,\mathtt{to}\, y.\, P) : \rho}$$

$$\frac{\Gamma \vDash_{\mathsf{p}} M_1 : \sigma \ldots \Gamma \vDash_{\mathsf{p}} M_n : \sigma \quad \Gamma, x : \sigma \vDash_{\mathsf{p}} N : \tau}{\Gamma \vDash_{\mathsf{p}} \mathsf{op}_\sigma[M_1, \ldots, M_n] \,\mathtt{to}\, x.\, N \equiv \mathsf{op}_\tau[M_1 \,\mathtt{to}\, x.\, N, \ldots, M_n \,\mathtt{to}\, x.\, N] : \tau}$$

2.4 Denotational semantics

We now recall the general programme of denotational semantics for the language in §2.2–2.3 in a category with sufficient structure [29,20,35]. Given an algebraic signature Op, a *monad model* is given by a category C with following data:

- a chosen cartesian closed structure, i.e. chosen finite products (including a terminal object 1), and for all objects A and B an object $[A \Rightarrow B]$ together with an evaluation morphism $\varepsilon : [A \Rightarrow B] \times A \to B$ such that for every $f : C \times A \to B$ there is a unique morphism $\lambda f : C \to [A \Rightarrow B]$ such that $f = \varepsilon \circ (\lambda f \times \mathsf{id}_A)$.
- a strong monad T on C, i.e. for each object A an object TA, and a morphism $\eta : A \to TA$, and for all objects A and B a morphism $\mathsf{str} : A \times TB \to T(A \times B)$, and for each morphism $f : A \to TB$ a morphism $f^* : TA \to TB$ (also called the Kleisli extension of f), satisfying appropriate conditions (e.g. [29]).
- for each operation $\mathsf{op} \in \mathsf{Op}$ with $\mathrm{ar}(\mathsf{op}) = n$, a natural transformation $\mathsf{T\text{-}op} : T(-)^n \to T(-)$ between functors $\mathsf{C} \to \mathsf{C}$.

We interpret the intermediate language in a monad model by interpreting types as objects and terms as morphisms. The interpretation of types as objects proceeds as follows: $[\![\mathtt{unit}]\!] \overset{\text{def}}{=} 1$, $[\![\tau_1 * \tau_2]\!] \overset{\text{def}}{=} [\![\tau_1]\!] \times [\![\tau_2]\!]$, $[\![\sigma \rightharpoonup \tau]\!] \overset{\text{def}}{=} [[\![\sigma]\!] \Rightarrow T[\![\tau]\!]]$. We interpret a context $(x_1 : \tau_1, \ldots, x_n : \tau_n)$ as an object too, as the product of the interpretations of its consituent types: $[\![(x_1 : \tau_1, \ldots, x_n : \tau_n)]\!] \overset{\text{def}}{=} [\![\tau_1]\!] \times \cdots \times [\![\tau_n]\!]$. That is, a context is interpreted as the object of environments for that context.

Value typing judgments $\Gamma \vDash_{\mathsf{v}} V : \tau$ are interpreted as morphisms $[\![V]\!]_{\mathsf{v}} : [\![\Gamma]\!] \to [\![\tau]\!]$, and producer typing judgments $\Gamma \vDash_{\mathsf{p}} M : \tau$ as morphisms $[\![M]\!]_{\mathsf{p}} : [\![\Gamma]\!] \longrightarrow T[\![\tau]\!]$.

These morphisms are defined by induction on the structure of derivations:

$$
\begin{array}{rclrcl}
[\![x]\!]_{\mathsf{v}} & \stackrel{\text{def}}{=} & \pi_x & [\![\mathtt{fn}\,x{:}\sigma \Rightarrow N]\!]_{\mathsf{v}} & \stackrel{\text{def}}{=} & \lambda[\![N]\!]_{\mathsf{p}} \\
[\![\mathtt{\#}_1\,V]\!]_{\mathsf{v}} & \stackrel{\text{def}}{=} & \pi_1 \circ [\![V]\!]_{\mathsf{v}} & [\![V\,W]\!]_{\mathsf{p}} & \stackrel{\text{def}}{=} & \varepsilon \circ \langle [\![V]\!]_{\mathsf{v}}, [\![W]\!]_{\mathsf{v}} \rangle \\
[\![\mathtt{\#}_2\,V]\!]_{\mathsf{v}} & \stackrel{\text{def}}{=} & \pi_2 \circ [\![V]\!]_{\mathsf{v}} & [\![\mathtt{return}\,V]\!]_{\mathsf{p}} & \stackrel{\text{def}}{=} & \eta \circ [\![V]\!]_{\mathsf{v}} \\
[\![\langle V, W\rangle]\!]_{\mathsf{v}} & \stackrel{\text{def}}{=} & \langle [\![V]\!]_{\mathsf{v}}, [\![W]\!]_{\mathsf{v}} \rangle & [\![M\,\mathtt{to}\,x.\,N]\!]_{\mathsf{p}} & \stackrel{\text{def}}{=} & [\![N]\!]^*_{\mathsf{p}} \circ \mathsf{str} \circ \langle \mathsf{id}, [\![M]\!]_{\mathsf{p}} \rangle \\
[\![\langle\rangle]\!]_{\mathsf{v}} & \stackrel{\text{def}}{=} & \langle\rangle & [\![\mathsf{op}_\tau[M_1, \ldots, M_n]]\!]_{\mathsf{p}} & \stackrel{\text{def}}{=} & \mathsf{T}\text{-}\mathsf{op} \circ \langle [\![M_1]\!]_{\mathsf{p}}, \ldots, [\![M_n]\!]_{\mathsf{p}} \rangle
\end{array}
$$

Proposition 2.1 (Soundness) *In any monad model:*
If $\Gamma \vDash_{\mathsf{v}} V \equiv W : \tau$ *then* $[\![V]\!]_{\mathsf{v}} = [\![W]\!]_{\mathsf{v}}$. *If* $\Gamma \vDash_{\mathsf{p}} M \equiv N : \tau$ *then* $[\![M]\!]_{\mathsf{p}} = [\![N]\!]_{\mathsf{p}}$.

For a simple example of a monad model, let C be the category Set of sets and functions between them. We can associate to any set A the least set $T(A)$ containing A and closed under the operations in Op. This yields a strong monad. The Eilenberg-Moore algebras for this monad can be understood as sets A that are equipped with a function $A^n \to A$ for each n-ary operation $\mathsf{op} \in \mathsf{Op}$. Unfortunately this set-theoretic model is not good enough for NBE, informally because it does not support reification at higher types. We build a model suitable for NBE in §3.2.

3 Normalization by evaluation

The general programme of NBE proceeds in three steps, following Section 1.1: identifying normal forms (§3.1), building a model that supports a denotational semantics (§3.2), and defining a reification from the model to the normal forms (§3.3).

3.1 Normal forms

The normal forms for our language are based on the η-long β-normal forms of simply typed lambda calculus. We mutually define judgements of normal values ($\vDash^{\mathsf{n}}_{\mathsf{v}}$), normal producers ($\vDash^{\mathsf{n}}_{\mathsf{p}}$), atomic values ($\vDash^{\mathsf{a}}_{\mathsf{v}}$) and atomic producers ($\vDash^{\mathsf{a}}_{\mathsf{p}}$).

$$
\frac{}{\Gamma, x : \tau, \Gamma' \vDash^{\mathsf{a}}_{\mathsf{v}} x : \tau}
\qquad
\frac{\Gamma \vDash^{\mathsf{n}}_{\mathsf{v}} V_1 : \tau_1 \quad \Gamma \vDash^{\mathsf{n}}_{\mathsf{v}} V_2 : \tau_2}{\Gamma \vDash^{\mathsf{n}}_{\mathsf{v}} \langle V_1, V_2 \rangle : \tau_1 * \tau_2}
\qquad
\frac{\Gamma, x : \sigma \vDash^{\mathsf{n}}_{\mathsf{p}} N : \tau}{\Gamma \vDash^{\mathsf{n}}_{\mathsf{v}} \mathtt{fn}\,x{:}\sigma \Rightarrow N : \sigma \rightharpoonup \tau}
$$

$$
\frac{}{\Gamma \vDash^{\mathsf{n}}_{\mathsf{v}} \langle\rangle : \mathtt{unit}}
\qquad
\frac{\Gamma \vDash^{\mathsf{a}}_{\mathsf{v}} V : \tau_1 * \tau_2}{\Gamma \vDash^{\mathsf{a}}_{\mathsf{v}} \mathtt{\#}_i\,V : \tau_i}
\qquad
\frac{\Gamma \vDash^{\mathsf{a}}_{\mathsf{v}} V : \sigma \rightharpoonup \tau \quad \Gamma \vDash^{\mathsf{n}}_{\mathsf{v}} W : \sigma}{\Gamma \vDash^{\mathsf{a}}_{\mathsf{p}} V\,W : \tau}
$$

$$
\frac{\Gamma \vDash^{\mathsf{n}}_{\mathsf{v}} V : \tau}{\Gamma \vDash^{\mathsf{n}}_{\mathsf{p}} \mathtt{return}\,V : \tau}
\qquad
\frac{\Gamma \vDash^{\mathsf{a}}_{\mathsf{p}} M : \sigma \quad \Gamma, x : \sigma \vDash^{\mathsf{n}}_{\mathsf{p}} N : \tau}{\Gamma \vDash^{\mathsf{n}}_{\mathsf{p}} M\,\mathtt{to}\,x.\,N : \tau}
\qquad
\frac{\Gamma \vDash^{\mathsf{n}}_{\mathsf{p}} M_1 : \tau \;\ldots\; \Gamma \vDash^{\mathsf{n}}_{\mathsf{p}} M_n : \tau}{\Gamma \vDash^{\mathsf{n}}_{\mathsf{p}} \mathsf{op}_\tau[M_1, \ldots, M_n] : \tau}
$$

The atomic judgements are an auxiliary notion that we use to define normal judgements. Informally, atomic judgements are built from destructors (projections, function application) and normal judgements are built from constructors (pairing, abstraction). The only thing that can be done with an atomic producer is to force its execution and substitute the result, using `to`. Atomic values can be substituted for variables without denormalizing a term.

3.2 A model of set theory with identifiers

Our NBE algorithm works over programs with free variables, that is, open programs. To accommodate this we build a model of set theory in which there is a 'set of identifiers' for each type. We do this categorically, using the presheaf construction, following [3,9,14]. (Nominal sets [32] are also related from a semantic perspective.)

A category of contexts and renamings: Let Ren be the category whose objects are contexts of our language: lists of types, informally annotated with variables. A morphism $(\sigma_1, \ldots, \sigma_m) \longrightarrow (\tau_1, \ldots, \tau_n)$ is given by a function $f : m \to n$ such that $\sigma_i = \tau_{f(i)}$ for $1 \leq i \leq m$. Composition of morphisms is composition of functions.

A category of presheaves: We will consider the category $\mathsf{Set}^{\mathsf{Ren}}$ of (covariant) presheaves. The objects are functors $\mathsf{Ren} \to \mathsf{Set}$, and the morphisms are natural transformations. We understand a functor $F : \mathsf{Ren} \to \mathsf{Set}$ as assigning to each context a set which may depend on the free variables in that context. The functorial action on morphisms accounts for renamings of variables.

A helpful perspective is to think of this category as a model of intuitionistic set theory (e.g. [23]). For any type τ there is a representable presheaf $\mathsf{Ren}(\tau, -)$ which may be thought of as a 'set of identifiers' labelled with the type τ. These identifiers are pure: they cannot be manipulated or compared.

The category $\mathsf{Set}^{\mathsf{Ren}}$ has products, sums and function spaces (e.g. [23, §III.6]).

- *products:* for presheaves $F_1, \ldots, F_n$ we let $(F_1 \times \cdots \times F_n)(\Gamma) = F_1(\Gamma) \times \cdots \times F_n(\Gamma)$.
- *coproducts:* let $(F_1 + \cdots + F_n)(\Gamma) = F_1(\Gamma) \uplus \cdots \uplus F_n(\Gamma)$.
- *cartesian closure:* for $F, G \in \mathsf{Set}^{\mathsf{Ren}}$, let $[F \Rightarrow G](\Gamma) = \mathsf{Set}^{\mathsf{Ren}}(\mathsf{Ren}(\Gamma, -) \times F, G)$.

Syntactic presheaves: For any type τ we have six presheaves $\mathsf{Ren} \to \mathsf{Set}$ built from the syntactic constructions in §2.2 and §3.1: presheaves of values (VTerms_τ), producers (PTerms_τ), normal values ($\mathsf{NVTerms}_\tau$), atomic values ($\mathsf{AVTerms}_\tau$), normal producers ($\mathsf{NPTerms}_\tau$) and atomic producers ($\mathsf{APTerms}_\tau$). For example, $\mathsf{VTerms}_\tau(\Gamma) \overset{\text{def}}{=} \{V \mid \Gamma \vDash\!\!\!\!\!\!\mathsf{v}\; V : \tau\}$. Presheaf actions are given by variable renaming.

A residualizing monad: The crux of our semantic analysis is our residualizing monad T on the presheaf category $\mathsf{Set}^{\mathsf{Ren}}$. We begin with an abstract description of it, and follow with a concrete inductive definition.

We briefly define a *residualizing algebra* to be a presheaf $F : \mathsf{Ren} \to \mathsf{Set}$ together with a natural transformation $F^n \to F$ for each n-ary operation in the signature Op, and also a natural transformation $\mathsf{APTerms}_\tau \times ([\mathsf{Ren}(\tau, -) \Rightarrow F]) \to F$ for each type τ. The algebraic structure from the signature interprets the effects in the signature, and the additional structure describes sequencing of effects with atomic producers. Recall that atomic producers are function calls involving free identifiers; their effects are undetermined. With suitably defined algebra homomorphisms, we arrive at a category which is monadic over $\mathsf{Set}^{\mathsf{Ren}}$. That is, the category of residualizing algebras is the category of Eilenberg-Moore algebras for a strong monad T on the category $\mathsf{Set}^{\mathsf{Ren}}$. (This follows from the 'crude monadicity theorem'.)

The monad T has the following concrete inductive description. Let $F : \mathsf{Ren} \to \mathsf{Set}$ be a presheaf. We define a new presheaf $TF : \mathsf{Ren} \to \mathsf{Set}$ so that the sets $TF(\Gamma)$

are the least satisfying the following rules:

$$\frac{d \in F(\Gamma)}{(\textsf{T-return}\, d) \in TF(\Gamma)} \qquad \frac{\Gamma \overset{\mathsf{a}}{\underset{\mathsf{p}}{\vdash}} M : \sigma \quad d \in TF(\Gamma, x{:}\sigma)}{(M\, \textsf{T-to}\, x.\, d) \ \in TF(\Gamma)} \qquad \frac{d_1 \in TF(\Gamma) \ \ldots \ d_n \in TF(\Gamma)}{\textsf{T-op}(d_1, \ldots, d_n) \ \in TF(\Gamma)}$$

The functorial action uses the action of F and the renaming of atomic producers. Note the tight correspondence between the residualizing monad and normal producers (§3.1): there is a natural isomorphism $\textsf{NPTerms}_\tau \cong T(\textsf{NVTerms}_\tau)$ (see also [21]). Another way to understand this monad is as the coproduct of the free monad generated by the algebraic signature $\textsf{Op}$ and the free monad generated by $\textsf{T-to}$ and $\textsf{T-return}$, as described by Ghani, Uustalu, Adámek and others [1,15].

Proposition 3.1 *The category* $\textsf{Set}^{\textsf{Ren}}$ *together with the residualizing monad* T *forms a monad model in the sense of* §*2.4.*

3.3 Reification and reflection

Recall that a nbe algorithm has two components: denotational semantics into the model, and reification back to normal forms.

We define reification as two families of natural transformations: ${}^{\mathsf{v}}{\downarrow}^{\tau \in \textsf{Ty}} \colon \llbracket \tau \rrbracket \to \textsf{NVTerms}_\tau$ and ${}^{\mathsf{p}}{\downarrow}^{\tau \in \textsf{Ty}} \colon T\llbracket \tau \rrbracket \to \textsf{NPTerms}_\tau$. To account for the contravariance at function types, the reification functions must be defined mutually with reflection functions, ${}^{\mathsf{v}}{\uparrow}^{\tau \in \textsf{Ty}} \colon \textsf{AVTerms}_\tau \to \llbracket \tau \rrbracket$ and ${}^{\mathsf{p}}{\uparrow}^{\tau \in \textsf{Ty}} \colon \textsf{APTerms}_\tau \to T\llbracket \tau \rrbracket$.

- ${}^{\mathsf{v}}{\downarrow}^{\tau} \colon \llbracket \tau \rrbracket \to \textsf{NVTerms}_\tau$ is defined by induction on the structure of types τ:

$$
\begin{aligned}
{}^{\mathsf{v}}{\downarrow}^{\texttt{unit}}_{\Gamma}\ d &\overset{\text{def}}{=} \langle\rangle \\
{}^{\mathsf{v}}{\downarrow}^{\tau_1 * \tau_2}_{\Gamma}\ d &\overset{\text{def}}{=} \langle {}^{\mathsf{v}}{\downarrow}^{\tau_1}_{\Gamma}\ (\pi_1\, d), {}^{\mathsf{v}}{\downarrow}^{\tau_2}_{\Gamma}\ (\pi_2\, d)\rangle \\
{}^{\mathsf{v}}{\downarrow}^{\sigma \rightharpoonup \tau}_{\Gamma}\ d &\overset{\text{def}}{=} \texttt{fn}\, x{:}\sigma \Rightarrow ({}^{\mathsf{p}}{\downarrow}^{\tau}_{\Gamma, x:\sigma}\ (\varepsilon\, \langle d, ({}^{\mathsf{v}}{\uparrow}^{\sigma}_{\Gamma, x:\sigma}\ x)\rangle))
\end{aligned}
$$

- ${}^{\mathsf{p}}{\downarrow}^{\tau} \colon T\llbracket \tau \rrbracket \to \textsf{NPTerms}_\tau$ is defined by induction on the structure of $T\llbracket \tau \rrbracket$:

$$
\begin{aligned}
{}^{\mathsf{p}}{\downarrow}^{\tau}_{\Gamma}\ (\textsf{T-return}\, d) &\overset{\text{def}}{=} \texttt{return}\, ({}^{\mathsf{v}}{\downarrow}^{\tau}_{\Gamma}\ d) \\
{}^{\mathsf{p}}{\downarrow}^{\tau}_{\Gamma}\ (M\, \textsf{T-to}\, x.\, d) &\overset{\text{def}}{=} M\, \textsf{to}\, x.\, ({}^{\mathsf{p}}{\downarrow}^{\tau}_{\Gamma, x:\sigma}\ d) \\
{}^{\mathsf{p}}{\downarrow}^{\tau}_{\Gamma}\ (\textsf{T-op}(d_1, \ldots, d_n)) &\overset{\text{def}}{=} \textsf{op}_\tau[{}^{\mathsf{p}}{\downarrow}^{\tau}_{\Gamma}\ d_1, \ldots, {}^{\mathsf{p}}{\downarrow}^{\tau}_{\Gamma}\ d_n]
\end{aligned}
$$

(Notice, $({}^{\mathsf{p}}{\downarrow}^{\tau})$ is derived from the natural isomorphism $\textsf{NPTerms}_\tau \cong T(\textsf{NVTerms}_\tau)$.)

- ${}^{\mathsf{v}}{\uparrow}^{\tau} \colon \textsf{AVTerms}_\tau \to \llbracket \tau \rrbracket$ is defined by induction on types τ:

$$
\begin{aligned}
{}^{\mathsf{v}}{\uparrow}^{\texttt{unit}}_{\Gamma}\ V &\overset{\text{def}}{=} \langle\rangle \\
{}^{\mathsf{v}}{\uparrow}^{\sigma \rightharpoonup \tau}_{\Gamma}\ V &\overset{\text{def}}{=} \lambda d.\, {}^{\mathsf{p}}{\uparrow}^{\tau}_{\Gamma, x:\sigma}\ (V\, ({}^{\mathsf{v}}{\downarrow}^{\sigma}_{\Gamma, x:\sigma}\ d)) \\
{}^{\mathsf{v}}{\uparrow}^{\tau_1 * \tau_2}_{\Gamma}\ V &\overset{\text{def}}{=} \langle {}^{\mathsf{v}}{\uparrow}^{\tau_1}_{\Gamma}\ (\pi_1\, V), {}^{\mathsf{v}}{\uparrow}^{\tau_2}_{\Gamma}\ (\pi_2\, V)\rangle
\end{aligned}
$$

- ${}^{\mathsf{p}}{\uparrow}^{\tau \in \textsf{Ty}} \colon \textsf{APTerms}_\tau \to T\llbracket \tau \rrbracket$ is defined by ${}^{\mathsf{p}}{\uparrow}^{\sigma}_{\Gamma}\ M \overset{\text{def}}{=} M\, \textsf{T-to}\, x.\, (\textsf{T-return}\, ({}^{\mathsf{v}}{\uparrow}^{\sigma}_{\Gamma, x:\tau}\ x))$.

Since variables are atomic values, the reflection morphisms allow us to map from the object of identifiers $\mathsf{Ren}(\tau, -)$ into the semantic domain $[\![\tau]\!]$, via the composite $\mathsf{Ren}(\tau, -) \longrightarrow \mathsf{AVTerms}_\tau \xrightarrow{{}^{\mathsf{v}}{\uparrow}^{\tau}} [\![\tau]\!]$.

3.4 Summary of the normalization algorithm

We now combine the denotational semantics with reification to build a normalization algorithm.

Any context $\Gamma = (x_1 : \tau_1 \ldots x_n : \tau_n)$ has an environment $\mathsf{id\text{-}env}_\Gamma$ (in the set $[\![\Gamma]\!]_\Gamma$) in which variables are interpreted as identifiers: let $\mathsf{id\text{-}env}_\Gamma \stackrel{\text{def}}{=} \langle {}^{\mathsf{v}}{\uparrow}^{\tau_1}_{\Gamma}\ x_1, \ldots, {}^{\mathsf{v}}{\uparrow}^{\tau_n}_{\Gamma}\ x_n \rangle$.

The normal form of a value judgement $\Gamma \vdash_{\mathsf{v}} V : \tau$ is found by reifying the interpretation $[\![V]\!]_{\mathsf{v}} : [\![\Gamma]\!] \to [\![\tau]\!]$ in the environment $\mathsf{id\text{-}env}_\Gamma$. Similarly the normal form of a producer judgement $\Gamma \vdash_{\mathsf{p}} M : \tau$ is found by reifying the interpretation $[\![M]\!]_{\mathsf{p}} : [\![\Gamma]\!] \to T[\![\tau]\!]$ in the environment $\mathsf{id\text{-}env}_\Gamma$:

$$\mathsf{nf}(V) \stackrel{\text{def}}{=} {}^{\mathsf{v}}{\downarrow}^{\tau}_{\Gamma}\ ([\![V]\!]_{\mathsf{v}\Gamma}(\mathsf{id\text{-}env}_\Gamma)) \qquad\qquad \mathsf{nf}(M) \stackrel{\text{def}}{=} {}^{\mathsf{p}}{\downarrow}^{\tau}_{\Gamma}\ ([\![M]\!]_{\mathsf{p}\Gamma}(\mathsf{id\text{-}env}_\Gamma))$$

We establish correctness of this normalization algorithm in Theorem 4.1.

Our normalization algorithm is based on a purely semantic analysis. Another common method for normalization is based on exhaustively rewriting syntactic program terms to compute their normal forms. To perform rewriting, one considers the equations $\Gamma \vdash_{\mathsf{v}} V \equiv W : \tau$ and $\Gamma \vdash_{\mathsf{p}} M \equiv N : \tau$ as rewrite rules. Lindley and Stark [22] have studied normalization for Moggi's monadic metalanguage in this setting. They developed a $\top\top$-lifting based proof method by building on the strong normalization results for simply-typed lambda calculus based on reducibility candidates (see also [11]).

3.5 A note on implementation

The algorithm in this section reduces normalization for the programming language to evaluation in set theory. For this to be an effective procedure, we need to understand the 'category of sets' in a constructive way. We do this using Agda [30], an implementation of Martin-Löf's type theory [24]. The structure of our implementation and its correctness proofs closely follow the presentation in this paper.

4 Correctness of the algorithm

We now show that the normalization algorithm we defined in §3 is correct. Our proof has been formalized in Agda. Similarly to [14], the proof of correctness is divided into three main theorems.

Theorem 4.1

(i) Normalization respects equivalence.
If $\Gamma \vdash_{\mathsf{v}} V \equiv W : \tau$ *then* $\mathsf{nf}(V) = \mathsf{nf}(W)$. *If* $\Gamma \vdash_{\mathsf{p}} M \equiv N : \tau$ *then* $\mathsf{nf}(M) = \mathsf{nf}(N)$.

(ii) Normalization preserves normal forms.
If $\Gamma \vdash^{\mathsf{n}}_{\mathsf{v}} V : \tau$ *then* $\mathsf{nf}(V) = V$. *If* $\Gamma \vdash^{\mathsf{n}}_{\mathsf{p}} M : \tau$ *then* $\mathsf{nf}(M) = M$.

(iii) Terms are equivalent to their normal forms.
If $\Gamma \vDash_{\mathsf{v}} V : \tau$ *then* $\Gamma \vDash_{\mathsf{v}} V \equiv \mathsf{nf}(V) : \tau$. *If* $\Gamma \vDash_{\mathsf{p}} M : \tau$ *then* $\Gamma \vDash_{\mathsf{p}} M \equiv \mathsf{nf}(M) : \tau$.

Item (i) follows immediately from soundness of semantics (Prop. 2.1 and 3.1). Item (ii) is proved by induction on the derivations of normal values/producers. In the remainder of this section we outline a proof of item (iii) using logical relations.

4.1 Relating values and producers with their denotations

We begin by defining Kripke logical relations between values/producers and their denotations: $\underset{\mathsf{v}}{\lhd}{}^{\tau}_{\Gamma} \subseteq [\![\tau]\!](\Gamma) \times \mathsf{VTerms}_{\tau}(\Gamma)$ and $\underset{\mathsf{p}}{\lhd}{}^{\tau}_{\Gamma} \subseteq (T[\![\tau]\!])(\Gamma) \times \mathsf{PTerms}_{\tau}(\Gamma)$. We define them by induction: $\underset{\mathsf{v}}{\lhd}{}^{\tau}$ on the structure of τ, $\underset{\mathsf{p}}{\lhd}{}^{\tau}$ on the structure of T.

$$\begin{aligned}
d \underset{\mathsf{v}}{\lhd}{}^{\mathtt{unit}}_{\Gamma} V &\overset{\text{def}}{\Longleftrightarrow} \mathsf{true}\\
d \underset{\mathsf{v}}{\lhd}{}^{\tau_1 * \tau_2}_{\Gamma} V &\overset{\text{def}}{\Longleftrightarrow} (\pi_1\, d \underset{\mathsf{v}}{\lhd}{}^{\tau_1}_{\Gamma} \#_1\, V) \wedge (\pi_2\, d \underset{\mathsf{v}}{\lhd}{}^{\tau_2}_{\Gamma} \#_2\, V)\\
d \underset{\mathsf{v}}{\lhd}{}^{\sigma \rightharpoonup \tau}_{\Gamma} V &\overset{\text{def}}{\Longleftrightarrow} \forall f \in \mathsf{Ren}(\Gamma, \Gamma'). \forall d', V'.\\
&\qquad\qquad d' \underset{\mathsf{v}}{\lhd}{}^{\sigma}_{\Gamma} V' \implies \varepsilon\,([\![\sigma \rightharpoonup \tau]\!]_f\, d, d') \underset{\mathsf{p}}{\lhd}{}^{\tau}_{\Gamma'} ((V[f])\, V')\\
(\mathsf{T\text{-}return}\, d) \underset{\mathsf{p}}{\lhd}{}^{\tau}_{\Gamma} M &\overset{\text{def}}{\Longleftrightarrow} \exists V.\, \Gamma \vDash_{\mathsf{p}} M \equiv \mathtt{return}\, V : \tau \wedge d \underset{\mathsf{v}}{\lhd}{}^{\tau}_{\Gamma} V\\
(N\, \mathsf{T\text{-}to}\, x.\, d) \underset{\mathsf{p}}{\lhd}{}^{\tau}_{\Gamma} M &\overset{\text{def}}{\Longleftrightarrow} \exists P.\, \Gamma \vDash_{\mathsf{p}} M \equiv N\, \mathtt{to}\, x.\, P : \tau \wedge d \underset{\mathsf{p}}{\lhd}{}^{\tau}_{\Gamma, x :\, \sigma} P\\
(\mathsf{T\text{-}op}(d_1 \ldots d_n)) \underset{\mathsf{p}}{\lhd}{}^{\tau}_{\Gamma} M &\overset{\text{def}}{\Longleftrightarrow} \exists M_1 \ldots M_n \in \mathsf{PTerms}_{\tau}(\Gamma).\\
&\qquad\qquad \Gamma \vDash_{\mathsf{p}} M \equiv \mathsf{op}_{\tau}[M_1, \ldots, M_n] : \tau \wedge d_1 \underset{\mathsf{p}}{\lhd}{}^{\tau}_{\Gamma} M_1 \wedge \ldots \wedge d_n \underset{\mathsf{p}}{\lhd}{}^{\tau}_{\Gamma} M_n
\end{aligned}$$

Proposition 4.2 The logical relations are invariant under equivalence: *If* $d \underset{\mathsf{v}}{\lhd}{}^{\tau}_{\Gamma} V$ *and* $\Gamma \vDash_{\mathsf{v}} V \equiv W : \tau$ *then* $d \underset{\mathsf{v}}{\lhd}{}^{\tau}_{\Gamma} W$. *If* $d \underset{\mathsf{p}}{\lhd}{}^{\tau}_{\Gamma} M$ *and* $\Gamma \vDash_{\mathsf{p}} M \equiv N : \tau$ *then* $d \underset{\mathsf{p}}{\lhd}{}^{\tau}_{\Gamma} N$.

Proposition 4.3 The logical relations are subobjects in $\mathsf{Set}^{\mathsf{Ren}}$. *For* $f \in \mathsf{Ren}(\Gamma, \Gamma')$*: If* $d \underset{\mathsf{v}}{\lhd}{}^{\tau}_{\Gamma} V$ *then* $[\![\tau]\!]_f(d) \underset{\mathsf{v}}{\lhd}{}^{\tau}_{\Gamma'} V[f]$. *If* $d \underset{\mathsf{p}}{\lhd}{}^{\tau}_{\Gamma} M$ *then* $(T[\![\tau]\!])_f(d) \underset{\mathsf{p}}{\lhd}{}^{\tau}_{\Gamma'} M[f]$.

Proposition 4.4 The logical relations interact well with reification and reflection.

(i) *If* $d \underset{\mathsf{v}}{\lhd}{}^{\tau}_{\Gamma} V$ *then* $\Gamma \vDash_{\mathsf{v}} ({}^{\mathsf{v}}{\downarrow}^{\tau}_{\Gamma}\, d) \equiv V : \tau$. *If* $d \underset{\mathsf{p}}{\lhd}{}^{\tau}_{\Gamma} M$ *then* $\Gamma \vDash_{\mathsf{p}} ({}^{\mathsf{p}}{\downarrow}^{\tau}_{\Gamma}\, d) \equiv M : \tau$.

(ii) *If* $\Gamma \vDash^{\mathsf{a}}_{\mathsf{v}} V : \tau$ *then* $({}^{\mathsf{v}}{\uparrow}^{\tau}_{\Gamma}\, V) \underset{\mathsf{v}}{\lhd}{}^{\tau}_{\Gamma} V$. *If* $\Gamma \vDash^{\mathsf{a}}_{\mathsf{p}} M : \tau$ *then* $({}^{\mathsf{p}}{\uparrow}^{\tau}_{\Gamma}\, M) \underset{\mathsf{p}}{\lhd}{}^{\tau}_{\Gamma} M$.

We extend logical relations to environments and simultaneous substitutions. For any context $\Gamma = (x_1 : \tau_1, \ldots, x_n : \tau_n)$, we let $\mathsf{Sub}_{\Gamma} \overset{\text{def}}{=} \mathsf{VTerms}_{\tau_1} \times \cdots \times \mathsf{VTerms}_{\tau_n}$. An element of Sub_{Γ} determines the substitution of a term for each variable in Γ. Given a judgement $\Gamma \vDash_{\mathsf{v}} V : \tau$, let $V[-] : \mathsf{Sub}_{\Gamma} \to \mathsf{VTerms}_{\tau}$ be defined by substitution. Similarly, given a producer $\Gamma \vDash_{\mathsf{p}} M : \tau$, we define $M[-] : \mathsf{Sub}_{\Gamma} \to \mathsf{PTerms}_{\tau}$ by substitution. We now define $\underset{\mathsf{v}}{\lhd}{}^{\Gamma} \subseteq [\![\Gamma]\!] \times \mathsf{Sub}_{\Gamma}$ as $e \underset{\mathsf{v}}{\lhd}{}^{\Gamma}_{\Gamma'} \rho \overset{\text{def}}{\Longleftrightarrow} \forall (x : \tau) \in \Gamma.\, (e\, x) \underset{\mathsf{v}}{\lhd}{}^{\tau}_{\Gamma'} (\rho\, x)$.

Proposition 4.5 (Fundamental lemma of logical relations) *If* $\Gamma \vDash_{\mathsf{v}} V : \tau$ *and* $e \underset{\mathsf{v}}{\lhd}{}^{\Gamma}_{\Gamma'} \rho$ *then* $([\![V]\!]_{\mathsf{v}}\, e) \underset{\mathsf{v}}{\lhd}{}^{\tau}_{\Gamma'} V[\rho]$. *If* $\Gamma \vDash_{\mathsf{p}} M : \tau$ *and* $e \underset{\mathsf{v}}{\lhd}{}^{\Gamma}_{\Gamma'} \rho$ *then* $([\![M]\!]_{\mathsf{p}}\, e) \underset{\mathsf{p}}{\lhd}{}^{\tau}_{\Gamma'} M[\rho]$.

4.2 Proof of Theorem 4.1(iii)

We use the logical relations to show that terms are equivalent to their normal forms.

Suppose $\Gamma \vDash_{\mathsf{v}} V : \tau$. We will show that $\Gamma \vDash_{\mathsf{v}} V \equiv \mathsf{nf}(V) : \tau$. (Recall that $\mathsf{nf}(V) \overset{\text{def}}{=} {}^{\mathsf{v}}{\downarrow}^{\tau}_{\Gamma}\ ([\![V]\!]_{\mathsf{v}\Gamma}(\mathsf{id\text{-}env}_{\Gamma}))$.) Using Prop. 4.4(ii), we deduce that identity environments and substitutions are related by $\mathrel{{}_{\mathsf{v}}{\lhd}}^{\Gamma}_{\Gamma'}$. By Prop. 4.5, $([\![V]\!]_{\mathsf{v}}\ \mathsf{id\text{-}env}) \mathrel{{}_{\mathsf{v}}{\lhd}}^{\tau}_{\Gamma} V$. From Prop. 4.4(i) we conclude $\Gamma \vDash_{\mathsf{v}} V \equiv \mathsf{nf}(V) : \tau$, as required. The case for producers is similar.

5 Equations and effects

The normalization process described in the previous sections is with respect to the equations in §2.3. We now discuss how to accommodate equations between effect terms.

5.1 Equations on effects

For a first example, the signature for non-determinism ($\oplus$) is usually considered together with the semilattice equations $x \oplus x = x$, $x \oplus y = y \oplus x$, $x \oplus (y \oplus z) = (x \oplus y) \oplus z$. To capture this in our language, we extend the equality for producers ($\Gamma \vDash_{\mathsf{p}} M \equiv N : \tau$, §2.3) by including these three equations at each type τ:

$$\frac{\Gamma \vDash_{\mathsf{p}} M : \tau}{\Gamma \vDash_{\mathsf{p}} M {\oplus} M \equiv M : \tau} \qquad \frac{\Gamma \vDash_{\mathsf{p}} M : \tau \quad \Gamma \vDash_{\mathsf{p}} N : \tau}{\Gamma \vDash_{\mathsf{p}} M {\oplus} N \equiv N {\oplus} M : \tau} \qquad \frac{\Gamma \vDash_{\mathsf{p}} M : \tau \quad \Gamma \vDash_{\mathsf{p}} N : \tau \quad \Gamma \vDash_{\mathsf{p}} P : \tau}{\Gamma \vDash_{\mathsf{p}} M {\oplus} (N {\oplus} P) \equiv (M {\oplus} N) {\oplus} P : \tau}$$

We also define equivalence relations on normal forms in a similar way:

$$\frac{\Gamma \vDash^{\mathsf{n}}_{\mathsf{p}} M : \tau}{\Gamma \vDash^{\mathsf{n}}_{\mathsf{p}} M {\oplus} M \equiv M : \tau} \qquad \frac{\Gamma \vDash^{\mathsf{n}}_{\mathsf{p}} M : \tau \quad \Gamma \vDash_{\mathsf{p}} N : \tau}{\Gamma \vDash^{\mathsf{n}}_{\mathsf{p}} M {\oplus} N \equiv N {\oplus} M : \tau} \qquad \frac{\Gamma \vDash^{\mathsf{n}}_{\mathsf{p}} M : \tau \quad \Gamma \vDash^{\mathsf{n}}_{\mathsf{p}} N : \tau \quad \Gamma \vDash^{\mathsf{n}}_{\mathsf{p}} P : \tau}{\Gamma \vDash^{\mathsf{n}}_{\mathsf{p}} M {\oplus} (N {\oplus} P) \equiv (M {\oplus} N) {\oplus} P : \tau}$$

Our NBE algorithm (§3) respects these equations:

Theorem 5.1

(i) *If* $\Gamma \vDash_{\mathsf{x}} V \equiv W : \tau$ *then* $\Gamma \vDash^{\mathsf{n}}_{\mathsf{x}} \mathsf{nf}(V) \equiv \mathsf{nf}(W) : \tau$, *for* $\mathsf{x} \in \{\mathsf{v}, \mathsf{p}\}$.

(ii) *If* $\Gamma \vDash^{\mathsf{n}}_{\mathsf{x}} V : \tau$ *then* $\mathsf{nf}(V) = V$, *for* $\mathsf{x} \in \{\mathsf{v}, \mathsf{p}\}$.

(iii) *If* $\Gamma \vDash_{\mathsf{x}} V : \tau$ *then* $\Gamma \vDash_{\mathsf{x}} V \equiv \mathsf{nf}(V) : \tau$, *for* $\mathsf{x} \in \{\mathsf{v}, \mathsf{p}\}$.

Although we do not have to change the NBE algorithm to respect the equivalence relations, we have to refine the residualizing model to establish correctness (Theorem 5.1). From a semantic perspective, we change the notion of residualizing algebra (§3.2), requiring that a residualizing algebra satisfies the semilattice equations. This gives us a different residualizing monad, which is a quotient of the monad in §3.2, so that we have an isomorphism $(\mathsf{NPTerms}_{\tau}/_{\equiv}) \cong T(\mathsf{NVTerms}_{\tau})$.

From the perspective of implementation, however, the types of Agda are intensional and they do not permit quotients by equivalence relations. To remedy this we revisit the semantic framework. We understand a 'set' as an Agda type equipped with a partial equivalence relation $\approx$ (PER: symmetric, transitive relation), following Cubric, Dybjer, Scott [9] and Pitts [33, §C.1]. For example, the type of functions $[X \Rightarrow Y]$ is equipped with the following PER: $f \approx_{X \to Y} g$ iff $\forall x, x' : X.\ x \approx_X x' \implies f(x) \approx_Y g(x')$. We are led to redo category theory in this setting, so that a 'hom-set' is actually a type equipped with a PER. For more details, see [9] or our Agda implementation.

There is nothing specific about semilattices in our analysis. In general, we accommodate equations on effects using the PER on the residualizing monad. Also importantly, the PERs are not visible in the constructions of the normalization algorithm. They only play a role in the formalization of the correctness argument (Theorem 5.1).

We mention in passing an alternative way to arrive at a suitable model to accommodate equations on effect terms: the setoid construction [7]. A setoid is a type equipped with an equivalence relation (that is also reflexive: $\forall x.\, x{\approx}x$). The setoid model has a different cartesian closed structure: the setoid of functions between given setoids X and Y is $\{f : X \to Y \mid x \approx_X x' \implies f(x) \approx_Y f(x')\}$. (This is roughly the same as the domain of the PER.) In a proof-relevant system like Agda, a setoid-based implementation of the normalization algorithm would be littered with proof witnesses for all inhabitants of function types. Although the setoid model is well behaved in many ways, the PER construction is better for our purposes because it yields an algorithm that is not complicated by proof obligations.

5.2 Normalization of effects

In the previous section we only identified normal forms up-to the equations on effect terms. In specific situations we can do better. For example, consider the signature for a one-bit memory cell: $\mathsf{Op} \overset{\text{def}}{=} \{\mathsf{lookup}, \mathsf{update}_0, \mathsf{update}_1\}$, $\mathrm{ar}(\mathsf{lookup}) \overset{\text{def}}{=} 2$, $\mathrm{ar}(\mathsf{update}_0) \overset{\text{def}}{=} 1$, $\mathrm{ar}(\mathsf{update}_1) \overset{\text{def}}{=} 1$, with the following equations [26,35]:

$$\begin{array}{cc} x = \mathsf{lookup}[\mathsf{update}_0[x], \mathsf{update}_1[x]] & \mathsf{update}_i[\mathsf{update}_j[x]] = \mathsf{update}_j[x] \\ \mathsf{update}_0[\mathsf{lookup}[x,y]] = \mathsf{update}_0[x] & \mathsf{update}_1[\mathsf{lookup}[x,y]] = \mathsf{update}_1[y] \end{array} \tag{1}$$

The idea is that $\mathsf{lookup}[M, N]$ is the program that reads the memory, continuing as M or N depending on the result, and $\mathsf{update}_i[M]$ writes i to the memory before continuing as M.

Rather than equipping the normal producers with a PER generated by these equations, we can instead represent effect terms directly in normal form, following Melliès [26]. We use an auxiliary judgement ($\Vdash^{\mathsf{n}'}_{\mathsf{p}}$).

$$\frac{\Gamma \Vdash^{\mathsf{n}'}_{\mathsf{p}} M : \tau \quad \Gamma \Vdash^{\mathsf{n}'}_{\mathsf{p}} N : \tau}{\Gamma \Vdash^{\mathsf{n}}_{\mathsf{p}} \mathsf{lookup}[M, N] : \tau} \quad \frac{\Gamma \Vdash^{\mathsf{n}'}_{\mathsf{p}} M : \tau \quad \Gamma \Vdash^{\mathsf{n}'}_{\mathsf{p}} N : \tau}{\Gamma \Vdash^{\mathsf{n}}_{\mathsf{p}} \mathsf{lookup}[\mathsf{update}_1[M], N] : \tau} \quad \frac{\Gamma \Vdash^{\mathsf{n}'}_{\mathsf{p}} M : \tau \quad \Gamma \Vdash^{\mathsf{n}'}_{\mathsf{p}} N : \tau}{\Gamma \Vdash^{\mathsf{n}}_{\mathsf{p}} \mathsf{lookup}[M, \mathsf{update}_0[N]] : \tau}$$

$$\frac{\Gamma \Vdash^{\mathsf{n}'}_{\mathsf{p}} M : \tau \quad \Gamma \Vdash^{\mathsf{n}'}_{\mathsf{p}} N : \tau}{\Gamma \Vdash^{\mathsf{n}}_{\mathsf{p}} \mathsf{lookup}[\mathsf{update}_1[M], \mathsf{update}_0[N]] : \tau} \quad \frac{\Gamma \Vdash^{\mathsf{n}}_{\mathsf{v}} V : \tau}{\Gamma \Vdash^{\mathsf{n}'}_{\mathsf{p}} \mathtt{return}\, V : \tau} \quad \frac{\Gamma \Vdash^{\mathsf{a}}_{\mathsf{p}} M : \sigma \quad \Gamma, x{:}\sigma \Vdash^{\mathsf{n}}_{\mathsf{p}} N : \tau}{\Gamma \Vdash^{\mathsf{n}'}_{\mathsf{p}} M \,\mathtt{to}\, x.\, N : \tau}$$

Recall that the residualizing monad is a coproduct of two monads. In the present case we can understand it as a coproduct of the residualizing monad for no effects (§3.2), and the one-bit-state monad $[\{0,1\} \Rightarrow ((-) \times \{0,1\})]$. Concretely, this coproduct of monads is the following least fixed point (following the definition in

[16]):

$$TF \;=\; \mu G.\ \Big[\{0,1\} \Rightarrow \big(\{0,1\} \times \big(F \;+\; \textstyle\sum_\tau (\mathsf{APTerms}_\tau \times [\mathsf{Ren}(\tau,-) \Rightarrow G])\big)\big)\Big]$$

In this monad the quotient by the equations (1) is made in the type, and a PER is not needed. Categorically speaking, this monad is isomorphic to the monad with a nontrivial PER. Concretely, however, this tailored monad provides a NBE algorithm that not only normalizes higher types, but also partially evaluates the imperative commands as much as possible. For illustration, consider the program (†) in the introduction, but with inp/out replaced by lookup/update. Rather than the normal form (‡), our algorithm also normalizes the effects, minimizing the number of writes:

$$\begin{aligned}&\mathsf{lookup}[h\,\langle\rangle\ \mathsf{to}\ x.\ \mathsf{lookup}[h\,\langle\rangle\ \mathsf{to}\ y.\ \mathsf{lookup}[\mathtt{return}\,\langle\rangle, \mathtt{return}\,\langle\rangle],\\ &\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{update}_0[h\,\langle\rangle\ \mathsf{to}\ y.\ \mathsf{lookup}[\mathtt{return}\,\langle\rangle, \mathtt{return}\,\langle\rangle]]],\\ &\qquad \mathtt{return}\,\langle\rangle].\end{aligned}$$

6 Remarks on extensions to the language

In this paper we have considered a restricted language with just enough features to demonstrate our contributions. While language features such as recursion and sum types are very important, they can be dealt with by using standard techniques from the literature. We briefly summarize the main ideas.

Recursion: Our NBE algorithm is guaranteed to terminate, because it is written in Agda. Nonetheless, realistic programming languages have the potential for non-termination. This leads us to the long-established connections between partial evaluation and NBE [10,12]. Roughly speaking, in a language with recursion, each sub-expression should be annotated with its 'binding time', to explain which parts of the program should be normalized at compile time (since they are somehow assumed to terminate) and which should not be touched until run time. Dybjer and Filinski [12,13] outline how to accommodate this in a monadic metalanguage.

Sum types: Most practical programming languages have sum types. For instance, we might have a type bit of bits with two constants $(0, 1)$ and following typing rule with equations:

$$\frac{\Gamma \vDash_{\mathsf{v}} V : \mathsf{bit} \quad \Gamma \vDash_{\mathsf{p}} M : \tau \quad \Gamma \vDash_{\mathsf{p}} N : \tau}{\Gamma \vDash_{\mathsf{p}} \mathsf{if}\ V\ \mathsf{then}\ M\ \mathsf{else}\ N : \tau} \qquad \begin{array}{c}\mathsf{if}\ i\ \mathsf{then}\ M_0\ \mathsf{else}\ M_1 \equiv M_i \quad (i = 0,1)\\ M \equiv \mathsf{if}\ a\ \mathsf{then}\ M[0/a]\ \mathsf{else}\ M[1/a]\end{array} \tag{2}$$

The semantic analysis based on presheaf categories has been extended to explain NBE with sum types for pure languages without computational effects [2,6]. Filinski [13] and Lindley [21] have discussed NBE for effectful languages with sums from a more pragmatic perspective. The languages they consider type `case` expressions as computations rather than as values, which allows them to use the residualizing monad to treat pattern-matching on atomic values.

Base types and local effects: Our residualizing monad is a monad on a presheaf category. Various authors use monads on presheaf categories to describe local effects and name generation, including local store [27,35,37], π-calculus [40], and logic programming [41]. The second author has recently developed a syntactic framework for these analyses, based on a generalized kind of algebraic theory [28,42], which can be accommodated in our semantic analysis (see also [27,37]). This framework allows us to move closer to the original source program in our introduction, as follows.

We can add to our grammar for types two abstract base types: a type chan of channels and a type bit of communication data. We can then modify our algebraic signature for input/output effects so that the operations take parameters from chan, specifying which channel to use for communication, and the input operation incorporates variable binding. This kind of signature is 'algebraic' in that it determines a monad on a presheaf category [41]. For input/output, we have this concrete syntax.

$$\frac{\Gamma \vdash_{\mathsf{v}} V : \mathsf{chan} \quad \Gamma, a : \mathsf{bit} \vdash_{\mathsf{p}} M : \tau}{\Gamma \vdash_{\mathsf{p}} \mathsf{inp}[V, a.\, M] : \tau} \qquad \frac{\Gamma \vdash_{\mathsf{v}} V : \mathsf{chan} \quad \Gamma \vdash_{\mathsf{v}} W : \mathsf{bit} \quad \Gamma \vdash_{\mathsf{p}} M : \tau}{\Gamma \vdash_{\mathsf{p}} \mathsf{out}[V, W, M] : \tau}$$

To allow manipulation of the data we add constants 0, 1 of type bit and also an operation $\mathsf{if\_then\_else\_}$ to our algebraic signature. In this way the typing rule in (2) arises from the algebraic signature of effects, not as an extra language construction. The equations for $\mathsf{if\_then\_else\_}$ (2) can be understood as part of the algebraic theory of the effects [42, §VC]. This suggests a new route to dealing with sum types in NBE, purely by using algebraic effects. We are currently experimenting with different implementations of the residualizing monad for this theory. We hope to recover a standard NBE algorithm for booleans [4] by implementing the monad carefully.

Handlers of algebraic effects: While algebraic effects give a general way for constructing impure computations, recent developments suggest that it is profitable to desconstruct computational effects. These 'effect handlers' generalize the idea of exception handlers to all algebraic effects. (See e.g. [34,38,18].)

To keep things simple, we consider the signature with one unary effect, op. We can add effect handlers for op to our language with the following term formation rule.

$$\frac{\Gamma \vdash_{\mathsf{p}} M : \sigma \quad \Gamma, x : \tau \vdash_{\mathsf{v}} H_{\mathsf{op}} : \tau \quad \Gamma, x : \sigma \vdash_{\mathsf{v}} H_{\mathtt{return}} : \tau}{\Gamma \vdash_{\mathsf{v}} \mathtt{handle}\, M\, \mathtt{with}\, \{\mathsf{op}(x) \Rightarrow H_{\mathsf{op}} \mid \mathtt{return}\,(x) \Rightarrow H_{\mathtt{return}}\} : \tau}$$

For an intuition, let $\mathsf{op}(M)$ be a computation that first 'beeps' and then continues as M. The expression $\mathtt{handle}\, M\, \mathtt{with}\, \{\mathsf{op}(x) \Rightarrow H_{\mathsf{op}} \mid \mathtt{return}\,(x) \Rightarrow H_{\mathtt{return}}\}$ then captures each of the beeps in M and replaces them with H_{op}. For instance, the expression

$$(\mathtt{handle}\, M\, \mathtt{with}\, \{\mathsf{op}(x) \Rightarrow \lambda\langle\rangle.\, \mathsf{op}(\mathsf{op}(x\langle\rangle)) \mid \mathtt{return}\,(x) \Rightarrow \lambda\langle\rangle.\, x\})\, \langle\rangle$$

replaces each 'beep' in M with two beeps.

Mathematically, handler expressions reify the idea that the type $(\langle\rangle \rightharpoonup \tau)$ is the free algebra on τ generated by the unary operation op. This intuition suggests the following equations: firstly, that the handlers are homomorphisms between unary algebras:

$$\begin{aligned}\Gamma \vdash_{\mathsf{v}} \mathtt{handle}\,(\mathtt{return}\,V)\,\mathtt{with}\,\{\mathsf{op}(x) \Rightarrow H_{\mathsf{op}} \,|\, \mathtt{return}\,(x) \Rightarrow H_{\mathtt{return}}\}\\ \equiv H_{\mathtt{return}}[V/x] : \tau\end{aligned}$$

$$\begin{aligned}\Gamma \vdash_{\mathsf{v}} \mathtt{handle}\,(\mathsf{op}(M))\,\mathtt{with}\,\{\mathsf{op}(x) \Rightarrow H_{\mathsf{op}} \,|\, \mathtt{return}\,(x) \Rightarrow H_{\mathtt{return}}\}\\ \equiv H_{\mathsf{op}}[(\mathtt{handle}\,M\,\mathtt{with}\,H)/x] : \tau\end{aligned}$$

and secondly, that the handlers provide unique mediating morphisms:

$$\frac{\Gamma, x : \mathtt{unit} \rightharpoonup \sigma \vdash_{\mathsf{v}} V[(\lambda\langle\rangle.\,\mathsf{op}[x\langle\rangle])/x] \equiv H_{\mathsf{op}}[(\lambda\langle\rangle.\,V)/y] : \tau}{\begin{aligned}&\Gamma, x : \mathtt{unit} \rightharpoonup \sigma \vdash_{\mathsf{v}}\\ &\quad V \equiv \mathtt{handle}\,(x\,\langle\rangle)\,\mathtt{with}\,\{\mathsf{op}(y) \Rightarrow H_{\mathsf{op}} \,|\, \mathtt{return}\,(z) \Rightarrow V[(\lambda\langle\rangle.\,\mathtt{return}\,z)/x]\} : \tau\end{aligned}}$$

However, we conjecture that this equational theory is undecidable. This conjecture is based on the observation that computations of type `unit` are essentially natural numbers (thinking of $\mathtt{return}\,\langle\rangle$ as zero and $\mathsf{op}(M)$ as the successor of M). Thus our system is close to Gödel's System T, in which equality is undecidable (assuming 'uniqueness of recursors': see [31]).

7 Summary

We have investigated normalization by evaluation for a language with higher types and computational effects. The effects are specified by an algebraic signature, so our algorithm works for any notion of computation that can be expressed this way.

A key contribution of our work is our clear and modular semantic analysis of normalization by evaluation. At the heart of our analysis is the *residualizing monad*.

- It is a monad on a presheaf category. Following Altenkirch, Cubric, Fiore and others [3,9,14], we use a presheaf category as an alternative to classical set theory because we need to normalize open terms. The presheaf category provides us with well-behaved 'sets of free identifiers', while supporting the standard approach to denotational semantics using cartesian closed categories.
- The monad is built in a principled and modular way, using the operations and equations in the algebraic theory that describes the computational effects, following the ideas of Plotkin, Power and others [35,26].
- In addition to algebraic operations, the monad also incorporates additional algebraic structure describing residualizing function calls, following Filinski [13].

Our normalization algorithm is implemented in the dependently typed language Agda, and also proved correct in Agda. To run our algorithm, we can naively think of sets as Agda types, but in the correctness proof we more properly understand sets as Agda types equipped with PERs, following [9].

References

[1] Adámek, J., S. Milius, N. Bowler and P. B. Levy, *Coproducts of monads on set*, in: *Proc. LICS* (2012).

[2] Altenkirch, T., P. Dybjer, M. Hofmann and P. Scott, *Normalization by evaluation for typed lambda calculus with coproducts*, in: *LICS'01*, Washington, DC, USA, 2001, pp. 303–310.

[3] Altenkirch, T., M. Hofmann and T. Streicher, *Categorical reconstruction of a reduction free normalization proof*, in: *CTCS'95*, 1995, pp. 182–199.

[4] Altenkirch, T. and T. Uustalu, *Normalization by evaluation for* $\lambda^{\rightarrow,2}$, in: *Proc. FLOPS'04*, 2004.

[5] Atkey, R., *A type checker that knows its monad from its elbow* (2011), http://bentnib.org/posts/2011-12-14-type-checker.html.

[6] Balat, V., R. Di Cosmo and M. Fiore, *Extensional normalisation and type-directed partial evaluation for typed lambda calculus with sums*, in: *POPL'04* (2004), pp. 64–76.

[7] Barthe, G., V. Capretta and O. Pons, *Setoids in type theory*, J. Funct. Program. **13** (2003), pp. 261–293.

[8] Berger, U. and H. Schwichtenberg, *An inverse of the evaluation functional for typed λ–calculus*, in: *Proc. LICS'91*, 1991, pp. 203–211.

[9] Cubric, D., P. Dybjer and P. J. Scott, *Normalization and the Yoneda embedding*, Math. Struct. Comput. Sci. **8** (1998), pp. 153–192.

[10] Danvy, O., *Type-directed partial evaluation*, in: *Proc. Partial Evaluation*, 1998, pp. 367–411.

[11] Doczkal, C. and J. Schwinghammer, *Formalizing a strong normalization proof for Moggi's computational metalanguage: a case study in Isabelle/Hol-nominal*, in: *Proc. LFMTP'09* (2009), pp. 57–63.

[12] Dybjer, P. and A. Filinski, *Normalization and partial evaluation*, in: *Proc. APPSEM 2000* (2002).

[13] Filinski, A., *Normalization by evaluation for the computational lambda-calculus*, in: *Proc. TLCA'01.*

[14] Fiore, M., *Semantic analysis of normalisation by evaluation for typed lambda calculus*, in: *Proc. PPDP'02*, 2002, pp. 26–37.

[15] Ghani, N. and T. Uustalu, *Coproducts of ideal monads*, ITA **38** (2004), pp. 321–342.

[16] Hyland, M., G. Plotkin and J. Power, *Combining effects: sum and tensor*, Theor. Comput. Sci. **357** (2006), pp. 70–99.

[17] Johann, P., A. Simpson and J. Voigtländer, *A generic operational metatheory for algebraic effects*, in: *Proc. LICS 2010*, 2010.

[18] Kammar, O., S. Lindley and N. Oury, *Handlers in action* (2013).

[19] Kammar, O. and G. D. Plotkin, *Algebraic foundations for effect-dependent optimisations*, in: *Proc. POPL 2012*, 2013, pp. 349–360.

[20] Levy, P. B., J. Power and H. Thielecke, *Modelling environments in call-by-value programming languages*, Information and Computation **185** (2003), pp. 182–210.

[21] Lindley, S., *Accumulating bindings*, in: O. Danvy, editor, *Informal proceedings of the 2009 Workshop on Normalization by Evaluation*, 2009, pp. 49–56.

[22] Lindley, S. and I. Stark, *Reducibility and* $\top\top$*-lifting for computation types*, in: *Proc. TLCA'05* (2005), pp. 262–277.

[23] Mac Lane, S. and I. Moerdijk, "Sheaves in geometry and logic: A First Introduction to Topos Theory," Springer-Verlag, 1992.

[24] Martin-Löf, P., *An intuitionistic theory of types, predicative part*, in: *Logic Colloquium 1973.*

[25] Martin-Löf, P., *About models for intuitionistic type theories and the notion of definitional equality*, in: S. Kanger, editor, *3rd Scandinavian Logic Symp.*, North-Holland, 1975 pp. 81–109.

[26] Melliès, P.-A., *Segal condition meets computational effects*, in: *Proc. LICS 2010*, 2010, pp. 150–159.

[27] Melliès, P.-A., *Local stores in string diagrams* (2011), http://tinyurl.com/mellies-itu-2011.

[28] Møgelberg, R. E. and S. Staton, *Linearly-used state in models of call-by-value*, in: *Proc. CALCO'11.*

[29] Moggi, E., *Notions of computation and monads*, Information and Computation **93** (1991), pp. 55–92.

[30] Norell, U., "Towards a Practical Programming Language Based on Dependent Type Theory," Ph.D. thesis, Chalmers University of Technology (2007).

[31] Okada, M. and P. Scott, *A note on rewriting theory for uniqueness of iteration*, Theory and Applications of Categories **6** (1999), pp. 47–64.

[32] Pitts, A. M., *Alpha-structural recursion and induction*, J. ACM **53** (2006), pp. 459–506.

[33] Pitts, A. M., *Structural recursion with locally scoped names*, J. Funct. Program. **21** (2011), pp. 235–286.

[34] Plotkin, G. and M. Pretnar, *Handlers of algebraic effects*, in: *Proc. ESOP 2009* (2009), pp. 80–94.

[35] Plotkin, G. D. and J. Power, *Notions of computation determine monads*, in: *Proc. FOSSACS'02* (2002).

[36] Plotkin, G. D. and J. Power, *Algebraic operations and generic effects*, Applied Categorical Structures **11** (2003), pp. 69–94.

[37] Power, J., *Indexed Lawvere theories for local state*, in: *Models, Logics and Higher-Dimensional Categories*, AMS, 2011 pp. 268–282.

[38] Pretnar, M., "The logic and handling of algebraic effects," Ph.D. thesis, University of Edinburgh (2010).

[39] Reppy, J. H., *Concurrent ML*, in: *Encyclopedia of Parallel Computing*, 2011 pp. 371–377.

[40] Stark, I., *Free-algebra models for the pi-calculus*, Theor. Comput. Sci. **390** (2008), pp. 248–270.

[41] Staton, S., *An algebraic presentation of predicate logic*, in: *Proc. FOSSACS 2013*, 2013, pp. 401–417.

[42] Staton, S., *Instances of computational effects*, in: *Proc. LICS 2013, to appear*, 2013, draft at http://www.cl.cam.ac.uk/~ss368/instances13.pdf.

MFPS 2013

On Concurrent Games with Payoff

Pierre Clairambault[1] and Glynn Winskel[2]

University of Cambridge

Abstract

The paper considers an extension of concurrent games with a **payoff**, *i.e.* a numerical value resulting from the interaction of two players. We extend a recent determinacy result on concurrent games [5] to a *value theorem*, *i.e.* a value that both players can get arbitrarily close to, whatever the behaviour of their opponent. This value is not reached in general, *i.e.* there is not always an optimal strategy for one of the players (there is for finite games). However when they exist, we show that optimal strategies are closed under composition, which opens up the possibility of computing optimal strategies for complex games compositionally from optimal strategies for their component games.

1 Introduction

Games are a well-established tool in mathematics, economics, logic, and of course computer science: in the latter, two-player games in particular are very widely used to model situations where an agent (*e.g.* a program) interacts with its environment (*e.g.* the user, the operating system). For instance, researchers in *game semantics* [9] have managed to build very precise (*fully abstract* [1,8]) models of higher-order programming languages with various computational effects. Another particularly rich line of work has been the application of game-theoretic tools for algorithmic and verification purposes: one expresses a desirable property of a system as a game, and reduces the satisfaction of this property to the existence of a "good" strategy for this game. Here, the meaning of "good" can be either qualitative (positions are winning or losing, with each player wanting to reach a winning position) or quantitative (positions have a given *payoff*, with both players trying to maximize their payoff). For these purposes, one generally wants the games considered to be *determined*: qualitatively, this means that one of the players necessarily has a winning strategy, and quantitatively that the game has a well-defined *value* that well-chosen strategies can reach or get arbitrarily close to. For this reason, the classes of games considered for these purposes generally enjoy *determinacy*: the most well-known such result is Martin's famous theorem [12] stating that for sequential, tree-like games whose

[1] Email: Pierre.Clairambault@cl.cam.ac.uk

[2] Email: Glynn.Winskel@cl.cam.ac.uk

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.com/locate/entcs

winning positions form a *Borel set*, one of the players must have a winning strategy. It is well-known that Martin's theorem generalizes to the quantitative setting if the game is *zero-sum*, *i.e.* in each position the payoffs of the two players sum to zero. In the last decade, there has been a growing interest in extensions of these games with *concurrency*. One very successful definition of (turn-based) concurrent games has been proposed by Henzinger, de Alfaro *et. al.* [3,4,7]: their games are based on Blackwell games [13], where at any point, the next state is decided by a function of parallel choices of both players. In these games, the *pure strategy* determinacy result of sequential games is weakened into a *mixed strategy* determinacy, where strategies are allowed to make probabilistic choices.

However in semantics, models of concurrent processes generally allow a more liberal, non turn-based form of concurrency. Starting with the work of Petri, many have come to advocate a view of concurrency based on *partial orders*, specifying the causal dependency between events – see [16] for an early summary of Petri's work and its relation with domain theory. Following this approach, several notions of concurrent games have been proposed as a basis for denotational semantics: in terms of closure operators [2] or asynchronous transition systems [15]. Recently, Winskel and Rideau introduced a more general setting for concurrent games [17]. It is based on the notion of *event structure* [18], a partial order of causal dependency on events with a consistency relation expressing nondeterministic choice. In the present paper, it is this framework that we will refer to by *concurrent games*. We showed in [5] that in this setting qualitative determinacy is satisfied for well-founded games meeting a structural condition called *race-freedom* expressing that moves of one player do not directly interfere with moves of the other. Here, we consider an extension of concurrent games with *zero-sum payoff*, and show a generalization of the qualitative determinacy result of [5] to a quantitative one. As the reader will see this is not a trivial exercise and requires a much finer analysis than for the qualitative case.

Note that we obtain *pure strategy* determinacy – our strategies do not make probabilistic choices, although they can act non-deterministically. There is an apparent contradiction with the line of work based on Blackwell games mentioned above, since they only have mixed strategy determinacy. This is due to a crucial difference between the two settings: in our games, no *fairness* assumption is made and strategies can legitimately choose *not to play*, possibly resulting in a deadlock if both strategies choose to do so. We argue that this is a desirable property, since very often in computer science we have to deal with systems that might not terminate. However from the game theory perspective, this implies that Blackwell games are *not* instances of our zero-sum concurrent games. (They *do* fit into our general framework, since fairness can be expressed by non zero-sum payoff by setting both players to be losing at incomplete positions.)

We also investigate quantitative features with respect to the *compositional* structure of concurrent games. In sequential games, strategies can be composed using a form of parallel composition and a hiding operation to make internal play invisible. This fact (first remarked on by Conway and emphasised by Joyal [10] in his analysis of Conway's work [6]) is seldom used in economics and algorithmics. However, it is at the very heart of game semantics, the compositional analysis of programs and programming languages in terms of games and strategies. Our concurrent games are compositional; in fact, the main result of [17] was to define and characterise strategies for which composition behaves well (*i.e.* is associative, and has identities). Not only is compositionality a prerequisite for building denotational models of programming languages (as they organize naturally as *categories*, see *e.g.* [11]), but it is also a very successful general approach for proving properties of complex programs. Adapting the earlier work on concurrent strategies, we show here that optimal strategies are stable under composition, thus building a bicategory of optimal strategies. This is a significant step towards a compositional analysis of optimal strategies: instead of modeling complex behaviours as payoff functions and then computing values and optimal strategies, construct complex optimal strategies by composition from elementary ones. Extensions with payoff should also prove useful for purely semantic purposes: pay-off is a powerful notion that allows us to express familiar *winning strategies* – as strategies of positive value – as well as more arcane game-theoretic notions, such as *well-bracketing* [14].

Outline.

In Section 2, we recall the framework of concurrent games originally presented in [17]. In Section 3, we show how to enrich these concurrent games with payoff and introduce the notion of value of games and strategies. In Section 4, we prove the main result of our paper, the value theorem. Finally in Section 5, we investigate the compositional aspects of payoff games; in particular we show that optimal strategies are stable under composition and form a bicategory.

2 Preliminaries

2.1 Event structures

An *event structure* comprises $(E, \leq, \mathrm{Con})$, consisting of a set E, of *events* which are partially ordered by $\leq$, the *causal dependency relation*, and a nonempty *consistency relation* Con consisting of finite subsets of E, which satisfy

$$\begin{array}{l} \{e' \mid e' \leq e\} \text{ is finite for all } e \in E, \\ \{e\} \in \mathrm{Con} \text{ for all } e \in E, \\ Y \subseteq X \in \mathrm{Con} \Rightarrow Y \in \mathrm{Con}, \quad \text{and} \\ X \in \mathrm{Con} \;\&\; e \leq e' \in X \Rightarrow X \cup \{e\} \in \mathrm{Con} \end{array}$$

The *configurations*, $\mathcal{C}^\infty(E)$, of an event structure E consist of those subsets $x \subseteq E$ which are

Consistent: $\forall X \subseteq x.\ X$ is finite $\Rightarrow X \in \mathrm{Con}$, and

Down-closed: $\forall e, e'.\ e' \leq e \in x \Rightarrow e' \in x$.

Often we shall be concerned with just the finite configurations of an event structure. We write $\mathcal{C}(E)$ for the *finite* configurations of an event structure E.

Two events which are both consistent and incomparable w.r.t. causal dependency in an event structure are regarded as *concurrent*. In games the relation of *immediate* dependency $e \rightarrowtriangle e'$, meaning e and e' are distinct with $e \leq e'$ and no event in between, will play an important role. For $X \subseteq E$ we write $[X]$ for $\{e \in E \mid \exists e' \in X.\ e \leq e'\}$, the down-closure of X; note if $X \in \mathrm{Con}$, then $[X] \in \mathrm{Con}$ is a configuration and in particular each event e is associated with a *prime* configuration $[e]$.

Notation 1 Let E be an event structure. We use $x \mathrel{-\!\!\!\!\subset} y$ to mean y covers x in $\mathcal{C}^\infty(E)$, *i.e.* $x \subset y$ in $\mathcal{C}^\infty(E)$ with nothing in between, and $x \overset{e}{\mathrel{-\!\!\!\!\subset}} y$ to mean $x \cup \{e\} = y$ for $x, y \in \mathcal{C}^\infty(E)$ and event $e \notin x$. We use $x \overset{e}{\mathrel{-\!\!\!\!\subset}}$, expressing that event e is enabled at configuration x, when $x \overset{e}{\mathrel{-\!\!\!\!\subset}} y$ for some y.

Definition 2.1 Let E and E' be event structures. A *(partial) map* of event structures $f : E \to E'$ is a partial function on events $f : E \rightharpoonup E'$ such that for all $x \in \mathcal{C}(E)$ its direct image $fx \in \mathcal{C}(E')$ and $\forall e_1, e_2 \in x,\ f(e_1) = f(e_2)$ (with both defined) $\Rightarrow e_1 = e_2$.

Maps of event structures compose as partial functions, with identity maps given by identity functions. We will say the map is *total* if the function f is total.

Definition 2.2 [Process operations]

- **Products.** The category of event structures with partial maps has *products* $A \times B$ with projections Π_1 to A and Π_2 to B. The effect is to introduce arbitrary synchronisations between events of A and events of B in the manner of process algebra.
- **Restriction.** The restriction of an event structure E to a subset of events R, written $E \upharpoonright R$, is the event structure with events $E' = \{e \in E \mid [e] \subseteq R\}$ and causal dependency and consistency induced by E.

Using these two operations, we can obtain a notion of **synchronized composition**. Synchronized compositions play a central role in process algebra, in such seminal work as Milner's CCS and Hoare's CSP. Synchronized compositions of event structures A and B are obtained as restrictions $A \times B \upharpoonright R$. We obtain *pullbacks* as a special case. Let $f : A \to C$ and $g : B \to C$ be maps of event structures. Defining P to be $A \times B \upharpoonright \{p \in A \times B \mid f\Pi_1(p) = g\Pi_2(p)$ with both defined$\}$, we obtain a pullback square

$$\begin{tikzcd}
& P \arrow[dl, "\Pi_1"'] \arrow[dr, "\Pi_2"] \arrow[dd, phantom, "\vee" very near start] & \\
A \arrow[dr, "f"'] & & B \arrow[dl, "g"] \\
& C &
\end{tikzcd}$$

in the category of event structures. When f and g are total the same construction gives the pullback in the category of event structures with *total* maps.

Definition 2.3 [Projection] Let $(E, \leq, \mathrm{Con})$ be an event structure. Let $V \subseteq E$ be a subset of 'visible' events. Define the *projection* of E on V, to be $E{\downarrow}V =_{\mathrm{def}} (V, \leq_V, \mathrm{Con}_V)$, where $v \leq_V v'$ iff $v \leq v'$ & $v, v' \in V$ and $X \in \mathrm{Con}_V$ iff $X \in \mathrm{Con}$ & $X \subseteq V$.

2.2 Concurrent strategies

2.2.1 Event structures with polarity

Both a game and a strategy in a game are to be represented using event structures with polarity, which comprise (E, pol) where E is an event structure with a polarity function $pol : E \to \{+, -\}$ ascribing a polarity $+$ (Player) or $-$ (Opponent) to its events. The events correspond to (occurrences of) moves. Maps of event structures with polarity are maps of event structures which preserve polarities.

Definition 2.4 [Basic operations]

- **Dual.** The *dual*, $E^\perp$, of an event structure with polarity E comprises the same underlying event structure E but with a reversal of polarities.
- **Simple parallel composition.** Let A and B be event structures with polarity. The operation $A \| B$ simply juxtaposes disjoint copies of A and B, maintaining their causal dependency and specifying a finite subset of events as consistent if it restricts to consistent subsets of A and B. Polarities are unchanged.

All the constructions previously introduced for event structures generalize directly in the presence of polarities.

2.2.2 Pre-strategies

Let A be an event structure with polarity, thought of as a game; its events stand for the possible occurrences of moves of Player and Opponent and its causal dependency and consistency relations stand for the constraints imposed by the game. A pre-strategy represents a nondeterministic play of the game—all its moves are moves allowed by the game and obey the constraints of the game; the concept will later be refined to that of *strategy.* A *pre-strategy in A* is defined to be a total map $\sigma : S \to A$ from an event structure with polarity S. Two pre-strategies $\sigma : S \to A$ and $\tau : T \to A$ in A will be essentially the same when they are isomorphic, *i.e.* there is an isomorphism $\theta : S \cong T$ such that $\sigma = \tau\theta$; then we write $\sigma \cong \tau$.

Let A and B be event structures with polarity. Following Joyal [10], a *pre-strategy from A to B* is a pre-strategy in $A^\perp \| B$, so a total map $\sigma : S \to A^\perp \| B$. We write $\sigma : A \mathrel{+\!\!\!\!\!\twoheadrightarrow} B$ to express that σ is a pre-strategy from A to B. Note that a pre-strategy σ *in* a game A, *e.g.* $\sigma : S \to A$, coincides with a pre-strategy *from* the empty game $\emptyset$ *to* the game A, *i.e.* $\sigma : \emptyset \mathrel{+\!\!\!\!\!\twoheadrightarrow} A$.

2.2.3 Composing pre-strategies

We present the composition of pre-strategies via pullbacks. Given two pre-strategies $\sigma : S \to A^\perp \| B$ and $\tau : T \to B^\perp \| C$, ignoring polarities we can consider the maps on the underlying event structures, *viz.* $\sigma : S \to A \| B$ and $\tau : T \to B \| C$. Viewed this way we can form the pullback in the category of event structures as shown below

$$\begin{tikzcd}
& A \parallel T \arrow[dr, "\mathrm{id}_A \| \tau"] & & \\
P \arrow[ur, "\Pi_2"] \arrow[dr, "\Pi_1"'] \arrow[r, phantom, "\rangle"] & & A \parallel B \parallel C \arrow[r] & A \parallel C \\
& S \parallel C \arrow[ur, "\sigma \| \mathrm{id}_C"'] & &
\end{tikzcd}$$

where the map $A\|B\|C \to A\|C$ is undefined on B and acts as identity on A and C. The partial map from P to $A\|C$ given by the diagram above (either way round the pullback square) factors as the composition of the partial map $P \to P \downarrow V$, where V is the set of events of P at which the map $P \to A\|C$ is defined, and a total map $P \downarrow V \to A\|C$. The resulting total map gives us the composition $\tau \odot \sigma : P \downarrow V \to A^\perp\|C$ once we reinstate polarities.

Identities w.r.t. composition are given by copy-cat strategies. Let A be an event structure with polarity. The copy-cat strategy from A to A is an instance of a pre-strategy, so a total map $\gamma_A : \mathbb{C}\!\mathbb{C}_A \to A^\perp\|A$. It describes a concurrent strategy based on the idea that Player moves, of positive polarity, always copy previous corresponding moves of Opponent, of negative polarity. For $c \in A^\perp\|A$ we use $\overline{c}$ to mean the corresponding copy of c, of opposite polarity, in the alternative component. Define $\mathbb{C}\!\mathbb{C}_A$ to comprise the event structure with polarity $A^\perp\|A$ together with the extra causal dependencies generated by $\overline{c} \leq_{\mathbb{C}\!\mathbb{C}_A} c$ for all events c with $pol_{A^\perp\|A}(c) = +$. The *copy-cat* pre-strategy $\gamma_A : A \mathrel{\rightarrow\!\!\!\!\!+\;} A$ is defined to be the map $\gamma_A : \mathbb{C}\!\mathbb{C}_A \to A^\perp\|A$ where γ_A acts as the identity function on the common set of events.

Interaction In this paper, we will be particularly interested in the results of the interaction between a strategy $\sigma : S \to B$ and a counter-strategy $\tau : T \to B^\perp$ in order to determine the resulting payoff. Unlike the composition $\tau \odot \sigma$ where the interaction of σ and τ are hidden, it is the status of the configurations in $\mathcal{C}^\infty(B)$ their full interaction induces which decides the resulting payoff. Ignoring polarities, we have total maps of event structures $\sigma : S \to B$ and $\tau : T \to B$. Form their pullback,

$$\begin{tikzcd}
 & P \arrow[dl, "\Pi_1"'] \arrow[dr, "\Pi_2"] \arrow[d, phantom, "\lrcorner" rotate=-45] & \\
S \arrow[dr, "\sigma"'] & & T \arrow[dl, "\tau"] \\
 & B\,, &
\end{tikzcd}$$

to obtain the event structure P resulting from the interaction of σ and τ. Because σ or τ may be nondeterministic there could be several maximal configurations in $\mathcal{C}^\infty(P)$. Define the set of results of the interaction of σ and τ to be

$$\langle \sigma, \tau \rangle =_{\mathrm{def}} \{\sigma \Pi_1 z \mid z \text{ is maximal in } \mathcal{C}^\infty(P)\}\,.$$

2.2.4 Strategies

The main result of [17] is that two conditions on pre-strategies, *receptivity* and *innocence*, are necessary and sufficient for copy-cat to behave as identity w.r.t. the composition of pre-strategies. Receptivity ensures an openness to all possible moves of Opponent. Innocence restricts the behaviour of Player; Player may only introduce new relations of immediate causality of the form $\ominus \rightarrowtriangle \oplus$ beyond those imposed by the game.

Definition 2.5 • **Receptivity.** A pre-strategy σ is *receptive* iff

$$\sigma x \overset{a}{\longrightarrow\!\!\!\subset} \ \&\ pol_A(a) = - \Rightarrow \exists! s \in S.\ x \overset{s}{\longrightarrow\!\!\!\subset} \ \&\ \sigma(s) = a\,.$$

- **Innocence.** A pre-strategy σ is *innocent* when it is both
  $+$*-innocent:* if $s \rightarrowtriangle s'$ & $pol(s) = +$ then $\sigma(s) \rightarrowtriangle \sigma(s')$, and

$-$*-innocent:* if $s \rightarrowtriangle s'$ & $pol(s') = -$ then $\sigma(s) \rightarrowtriangle \sigma(s')$.

A **strategy** is a receptive and innocent pre-strategy.

Theorem 2.6 (from [17]) *Let $\sigma : A \nrightarrow B$ be pre-strategy. Copy-cat behaves as identity w.r.t. composition, i.e. $\sigma \circ \gamma_A \cong \sigma$ and $\gamma_B \circ \sigma \cong \sigma$, iff σ is receptive and innocent. Copy-cat pre-strategies $\gamma_A : A \nrightarrow A$ are receptive and innocent.*

Theorem 2.6 motivated the definition of a *strategy* as a pre-strategy which is receptive and innocent. In fact, we obtain a bicategory, **Games**, in which the objects are event structures with polarity—the games, the arrows from A to B are strategies $\sigma : A \nrightarrow B$ and the 2-cells are maps of spans. The vertical composition of 2-cells is the usual composition of maps of spans. Horizontal composition is given by the composition of strategies $\odot$ (which extends to a functor on 2-cells via the universality of pullback).

3 Concurrent games with payoff

We begin the core of the paper, the treatment of payoff in concurrent games. $\overline{\mathbb{R}}$ denotes $\mathbb{R} \cup \{-\infty, +\infty\}$, the reals extended with a minimum and maximum.

Definition 3.1 A **concurrent game with payoff** is a triple $(A, \kappa_A^+, \kappa_A^-)$, where A is a concurrent game and $\kappa_A^+, \kappa_A^- : \mathcal{C}^\infty(A) \to \overline{\mathbb{R}}$ are **payoff functions**.

In all of this paper, we will only consider **zero-sum** concurrent games, *i.e.* for all $z \in \mathcal{C}^\infty(A)$, $\kappa_A^-(z) = -\kappa_A^+(z)$. It follows that our games with payoff will be described by a concurrent game and its payoff function $\kappa_A = \kappa_A^+ : \mathcal{C}^\infty(A) \to \overline{\mathbb{R}}$. We extend the usual constructions on concurrent games to games with payoff.

Definition 3.2 [Constructions]

- **Dual.** If A is a concurrent game with payoff, then the payoff function on $A^\perp$ is defined by $\kappa_{A^\perp}(x) = -\kappa_A(x)$, for $x \in \mathcal{C}^\infty(A^\perp)$.
- **Parallel composition.** If A, B are concurrent games with payoff, then the payoff function on $A \parallel B$ is defined by $\kappa_{A\parallel B}(x) = \kappa_A(x_1) + \kappa_A(x_2)$, where $x_1 \in \mathcal{C}^\infty(A)$ is the projection of x on A and $x_2 \in \mathcal{C}^\infty(B)$ is the projection of x on B.

We now turn to the definitions leading to the value of a game. Since games and strategies are nondeterministic, these definitions come in two variants: the *optimistic* describing the outcome of a game if all the nondeterministic choices are in favour of Player, and the *pessimistic* describing the dual case, when all of those choices are in favour of Opponent. One of the main result of the paper will be that for race-free well-founded games (to be defined below), the two corresponding notions of value coincide.

Definition 3.3 We define the **optimistic** ($\uparrow$) and **pessimistic** ($\downarrow$) **results** of an interaction, and **values** of a strategy and of a game, as follows. Here, σ is a strategy on A and τ is a counter-strategy (a strategy on $A^\perp$), and the notation $\sigma : A$ signifies a strategy $\sigma : S \to A$.

$$\begin{array}{rclcrcl}
r^{\uparrow}(\sigma,\tau) & = & \sup_{x\in\langle\sigma,\tau\rangle} \kappa_A(x) & \qquad & r^{\downarrow}(\sigma,\tau) & = & \inf_{x\in\langle\sigma,\tau\rangle} \kappa_A(x) \\
v^{\uparrow}(\sigma) & = & \inf_{\tau : A^{\perp}} r^{\uparrow}(\sigma,\tau) & & v^{\downarrow}(\sigma) & = & \inf_{\tau : A^{\perp}} r^{\downarrow}(\sigma,\tau) \\
v^{\uparrow}(A) & = & \sup_{\sigma : A} v^{\uparrow}(\sigma) & & v^{\downarrow}(A) & = & \sup_{\sigma : A} v^{\downarrow}(\sigma)
\end{array}$$

We say that a game A **has a value** if $v^{\uparrow}(A) = v^{\downarrow}(A) = -v^{\downarrow}(A^{\perp}) = -v^{\uparrow}(A^{\perp})$: the optimistic and pessimistic values coincide, and commute with $(-)^{\perp}$. The commutation with $(-)^{\perp}$ is a form of minimax property, since the order of quantification on strategies is reversed in $v(A)$ and $-v(A^{\perp})$, whereas the coincidence of the optimistic and pessimistic value deals with nondeterminism. Note that not all games have a value:

Example 3.4 Take the game $A = \ominus \sim\!\sim\!\sim \oplus$ with two events of opposite polarities conflicting with each other, along with $\kappa(\emptyset) = 0$, $\kappa(\{\oplus\}) = 1$ and $\kappa(\{\ominus\}) = -2$. Then it is easy to prove that $v^{\uparrow}(A) = 1$, $v^{\downarrow}(A) = -2$, $v^{\uparrow}(A^{\perp}) = 2$ and $v^{\downarrow}(A^{\perp}) = -1$.

The example above suggests a simple relationship between $v^{\downarrow}(A)$ and $v^{\uparrow}(A^{\perp})$ but this is not always the case. For example, consider the infinite game A comprising the event structure with polarity

$$\begin{tikzcd}
\ominus & \oplus_1 \arrow[r, -Triangle] & \oplus_2 \arrow[r, -Triangle] & \oplus_3 \arrow[r, -Triangle] & \cdots \arrow[r, -Triangle] & \oplus_n \arrow[r, -Triangle] & \cdots
\end{tikzcd}$$

where $\kappa(\emptyset) = 0$, $\kappa(\{\oplus_1, \ldots, \oplus_n\}) = n$, $\kappa(\{\oplus_1, \ldots, \oplus_n\} \cup \{\ominus\}) = -n$, $\kappa(\{\oplus_1, \ldots\}) = -\infty$ and $\kappa(\{\oplus_1, \ldots\} \cup \{\ominus\}) = +\infty$. Then one can check that the optimistic and pessimistic values coincide, in fact this is always the case when games do not have conflict. A direct analysis of the available strategies for Player and Opponent shows that $v(A) = 0$, whereas $v(A^{\perp}) = +\infty$.

The first example features a *race*, where both players compete for the same resource, whereas the second example is not *well-founded*: the game allows infinite configurations. These brings us to the two following notions, that will be crucial to get the value theorem.

Definition 3.5 A game A is **race-free** if for all $x \in \mathcal{C}(A)$ such that $x \overset{a}{-\!\!\!\subset}$ and $x \overset{a'}{-\!\!\!\subset}$ with $pol(a) = -$ and $pol(a') = +$, we have $x \cup \{a, a'\} \in \mathcal{C}(A)$.

A game A is **well-founded** if every configuration in $\mathcal{C}^{\infty}(A)$ is finite.

Definition 3.6 Let A be a concurrent game with payoff, and $x \in \mathcal{C}^{\infty}(A)$. Let A/x be the **residual of** A **after** x, comprising

- events, $\{a \in A \setminus x \mid x \cup [a]_A \in \mathcal{C}^{\infty}(A)\}$,
- consistency relation, $X \in \mathrm{Con}_{A/x} \Leftrightarrow X \subseteq_f A/x \ \&\ x \cup [X]_A \in \mathcal{C}^{\infty}(A)$,
- causal dependency, the restriction of that on A.

Define $\kappa_{A/x} : \mathcal{C}^{\infty}(A/x) \to \overline{\mathbb{R}}$ by taking $\kappa_{A/x}(y) = \kappa_A(x \cup y)$. Finally, define $(A, \kappa_A)/x = (A/x, \kappa_{A/x})$. When x is a singleton $\{a\}$, we shall generally write A/a instead of $A/\{a\}$. Finally, we will often write $v^{\uparrow}(x)$ (resp. $v^{\downarrow}(x)$) for $v^{\uparrow}(A/x)$ (resp. $v^{\downarrow}(A/x)$).

4 The value theorem

In this section, we prove the value theorem on concurrent games. The proof proceeds in two steps. First, we exhibit key constructions on strategies and the study the results of their interactions. This analysis will allow us to characterize the values of all positions of the game. Exploiting well-foundedness of the game, we will deduce the sought-for value theorem.

4.1 Constructions on strategies

In “glueing” strategies together it is helpful to assume that all the initial negative moves of the strategies are exactly the same, and indeed coincide with the initial negative moves of the game:

Lemma 4.1 *Let $\sigma : S \to A$ be a strategy, then there exists a strategy $\sigma' : S' \to A$ with $\sigma' \cong \sigma$, for which*

$$\forall s' \in S'.\ pol_{S'}[s']_{S'} = \{-\} \Rightarrow \sigma'(s') = s' \,. \qquad (\dagger)$$

Henceforth we will assume all strategies satisfy the property (†). In particular, its adoption facilitates the definition of a ‘sum’ of strategies in a game.

Proposition 4.2 *Let $\sigma_i : S_i \to A$, for $i \in I$, be strategies (assumed to satisfy (†)). W.l.og. we may assume that whenever indices $i, j \in I$ are distinct then so are those events of S_i and S_j which causally depend on a positive event (otherwise we could tag such events by their respective indices). Define S to be the event structure with events $\bigcup_{i\in I} S_i$, causal dependency $s \leq_S e'$ iff $s \leq_{S_i} e'$, for some $i \in I$, and consistency $X \in \mathrm{Con}_S$ iff $X \in \mathrm{Con}_{S_i}$, for some $i \in I$. Defining $\|_{i\in I}\, \sigma_i(s) = \sigma_i(s)$ if $s \in S_i$ yields a strategy $\|_{i\in I}\, \sigma_i : S \to A$.*

The next construction takes a strategy σ on a game A/a, where a is a minimal positive event of game A, and creates a strategy on A that starts by playing a, then resumes as σ.

Proposition 4.3 *Suppose A is a race-free game such that $\emptyset \overset{a}{{-}\!\!\!\subset}$ with $pol(a) = +$. Then for any strategy $\sigma : S \to A/a$, where w.l.o.g. $a \notin S$, there is a strategy $\mathrm{play}_a(\sigma) : S' \to A$: the event structure S' comprises*

- *events, $S \cup \{a\}$,*
- *causal dependency, that on S extended by $a \leq_{S'} s$, for $s \in S$, whenever $a \leq_A \sigma(s)$,*
- *with consistency, $X \in \mathrm{Con}_{S'}$ iff $X \cap S \in \mathrm{Con}_S$,*

and $\mathrm{play}_a(\sigma)(s) = \sigma(s)$, for $s \in S$, with $\mathrm{play}_a(\sigma)(a) = a$.

Given a strategy on σ on a residual game A/a, where a is an initial negative event of A, we can create a strategy on A that awaits a, then resumes as σ.

Proposition 4.4 *Suppose A is a game such that $\emptyset \overset{a}{{-}\!\!\!\subset}$ with $pol(a) = -$. Then for any strategy $\sigma : S \to A/a$, where w.l.o.g. $a \notin S$, there is a strategy $\mathrm{wait}_a(\sigma) : S' \to A$: the event structure S' comprises*

- *events, $S \cup A_-$, where $A_- =_{\mathrm{def}} \{a' \in A \mid pol_A[a']_A \subseteq \{-\}\}$,*

- *causal dependency, that on S and A_- extended by $a \leq_{S'} s$, for $s \in S$, whenever $a \leq_A \sigma(s)$ or $pol(s) = +$,*
- *with consistency, $X \in \mathrm{Con}_{S'}$ iff $X \cap S \in \mathrm{Con}_S$ & $\mathrm{wait}_a(\sigma)X \in \mathrm{Con}_A$,*

where $\mathrm{wait}_a(\sigma)(s')$ is defined to be $\sigma(s')$ if $s' \in S$, otherwise s'.

We will make use later of the following extension of the notion of residual from games to strategies:

Definition 4.5 Let $\sigma : S \to A$ be a strategy and $x \in \mathcal{C}^\infty(S)$. Define the function $\sigma/x : S/x \to A/\sigma x$ to be the restriction of σ. In the case where x is a singleton $\{s\}$, we shall generally write σ/s instead of $\sigma/\{s\}$.

Proposition 4.6 *For $\sigma : S \to A$ a strategy and $x \in \mathcal{C}^\infty(S)$, the function $\sigma/s : S/s \to A/\sigma(s)$ is a strategy.*

Let A be a game with payoff κ_A and $\sigma : S \to A$ and $\tau : T \to A^\perp$ be strategies. The set of values resulting from their interaction is given by $\{\kappa_A x \mid x \in \langle \sigma, \tau \rangle\}$, which we generally write as $\kappa\langle\sigma,\tau\rangle$ when the game is clear from the context. We use $\langle\sigma,\tau\rangle^+ =_{\mathrm{def}} \{x \in \langle\sigma,\tau\rangle \mid + \in pol\ x\}$ for the configurations in $\langle\sigma,\tau\rangle$ containing events of positive polarity. We will make crucial use of the following analysis of the interactions between strategies.

Lemma 4.7 *Let A be a well-founded race-free game with payoff. Let σ and σ_i, for $i \in I$, be strategies in A, and τ a strategy in $A^\perp$. Then,*

$$\begin{aligned} \kappa\langle\sigma,\tau\rangle &= \{-v \mid v \in \kappa\langle\tau,\sigma\rangle\} & \kappa\langle\mathrm{play}_a(\sigma),\tau\rangle &= \kappa\langle\sigma,\tau/a\rangle \\ \kappa\langle{\textstyle\parallel_{i\in I}}\,\sigma_i,\tau\rangle &\subseteq {\textstyle\bigcup_{i\in I}}\,\kappa\langle\sigma_i,\tau\rangle & \kappa\langle{\textstyle\parallel_{i\in I}}\,\sigma_i,\tau\rangle^+ &= {\textstyle\bigcup_{i\in I}}\,\kappa\langle\sigma_i,\tau\rangle^+ \\ \kappa\langle\mathrm{wait}_a(\sigma),\tau\rangle &\supseteq {\textstyle\bigcup_{t:\tau(t)=a}}\,\kappa\langle\sigma,\tau/t\rangle & \kappa\langle\mathrm{wait}_a(\sigma),\tau\rangle^+ &= {\textstyle\bigcup_{t:\tau(t)=a}}\,\kappa\langle\sigma,\tau/t\rangle^+ \end{aligned}$$

¿From this follow two important corollaries. Firstly, if a is an initial positive event of A we have $\kappa\langle\mathrm{play}_a(\sigma),\mathrm{wait}_a(\tau)\rangle = \kappa\langle\sigma,\tau\rangle$; two strategies, one playing a move and the other waiting for the move, synchronise. This immediately follows from the lemma above and the observation that $\mathrm{wait}_a(\sigma)/a = \sigma$. Secondly, the following additional construction will be crucial. For $(e_i)_{i\in I}$ the family of negative minimal events of A and strategies $\sigma_i : S_i \to A/e_i$, we define $\mathrm{case}_{i\in I}\sigma_i =_{\mathrm{def}} \parallel_{i\in I} \mathrm{wait}_{e_i}\sigma_i$. Roughly, this strategy waits for an input e_i and then proceeds as σ_i; though the full story is subtle as two distinct events e_i and e_j may be consistent with each other and the strategies σ_i and σ_j overlap. ¿From the lemma follows that for all $\tau : T \to A^\perp$ such that T has a minimal +-event

$$\kappa\langle\mathrm{case}_{i\in I}\sigma_i,\tau\rangle \subseteq \bigcup_{i\in I, t:\tau(t)=a_i} \kappa\langle\sigma_i,\tau/t\rangle \qquad \kappa\langle\mathrm{case}_{i\in I}\sigma_i,\tau\rangle^+ = \bigcup_{i\in I, t:\tau(t)=a_i} \kappa\langle\sigma_i,\tau/t\rangle^+$$

In Lemma 4.7 and the observation above, in all the cases where we have inclusions instead of equalities this is by necessity. For instance with the case construction above, a configuration in $\langle\sigma_i,\tau\rangle$, by definition a maximal configuration of the pullback of σ_i and τ, although it reappears as a configuration of the pullback of $\mathrm{case}_{i\in I}\sigma_i$ and τ, may no longer be maximal so fail to contribute to $\langle\mathrm{case}_{i\in I}\sigma_i,\tau\rangle$.

4.2 Values of these constructions

Lemma 4.8 *For any race-free well-founded game A, we have:*

$$v^{\uparrow}(\mathrm{play}_a(\sigma)) \leq v^{\uparrow}(\sigma) \qquad v^{\downarrow}(\mathrm{play}_a(\sigma)) \leq v^{\downarrow}(\sigma)$$
$$v^{\uparrow}(\sigma) \leq v^{\uparrow}(\sigma/a) \qquad v^{\downarrow}(\sigma) \leq v^{\downarrow}(\sigma/a)$$

Proof. Direct consequence of Lemma 4.7. □

Lemma 4.9 *Suppose A is race-free and well-founded and $\sigma : S \to A$ is a strategy with a minimal $+$-event. Let $(f_j)_{j\in J}$ be the family of minimal $+$-events of A. Then, $v^{\downarrow}(\sigma) \leq \sup_{j\in J} v^{\downarrow}(\{f_j\})$ and $v^{\uparrow}(\sigma) \leq \sup_{j\in J} v^{\uparrow}(\{f_j\})$.*

Proof. The pessimistic case follows from Lemma 4.7. *Optimistic case.* Suppose that the inequality is false, *i.e.* $\sup_{j\in J} v^{\uparrow}(\{f_j\}) < v^{\uparrow}(\sigma)$. This implies that there is $\alpha \in \mathbb{R}$ such that $\sup_{j\in J} v^{\uparrow}(\{f_j\}) < \alpha$ and $v^{\uparrow}(\sigma) > \alpha$. The first inequality implies $\forall j \in J,\ \forall \sigma' : A'/f_j,\ \exists \tau' : A^{\perp}/f_j,\ \forall z' \in \langle \sigma', \tau'\rangle,\ \kappa(z') < \alpha$, which is easily shown to imply

(1) $\forall (\sigma_k)_{k\in K},\ \exists (\tau_j)_{j\in\sigma K},\ \forall k \in K,\ \forall z' \in \langle \sigma_k, \tau_{\sigma k}\rangle,\ \kappa(z') < \alpha$

where K is the set of positive minimal events in S. Applying this property to the family of strategies obtained by $\sigma_k = \sigma/k$, we get a family of counter-strategies $(\tau_j)_{j\in\sigma K}$. We extend this family to J by setting τ_j to be the empty strategy (closed under receptivity) whenever $e_j \not\in \sigma K$. Thus, we get a family $(\tau_j)_{j\in J}$. Similarly, the second inequality implies that

$$\forall \tau : A^{\perp},\ \exists z \in \langle \sigma, \tau\rangle,\ \kappa(z) > \alpha\,.$$

Applied to $\tau = \mathrm{case}_{j\in J}\tau_j$, we get $z \in \langle \sigma, \mathrm{case}_{j\in J}\tau_j\rangle$ such that $\kappa(z) > \alpha$. By our observation on the interaction with case, there is $k_0 \in K$, and $z' \in \langle \sigma/k_0, \tau_{\sigma k_0}\rangle$ such that $\kappa(z') = \kappa(z) > \alpha$. However, applying (1) to k_0 also shows that $\kappa(z') < \alpha$, contradiction. Hence, the required inequality is true. □

Lemma 4.10 *Let A be a race-free well-founded game and $(e_i)_{i\in I}$ the family of its negative minimal events. Then,*

$$\min(\kappa(\emptyset), \inf_{i\in I} \sup_{\sigma:A/e_i} v^{\downarrow}(\sigma)) \leq v^{\downarrow}(A)$$
$$\min(\kappa(\emptyset), \inf_{i\in I} \sup_{\sigma:A/e_i} v^{\uparrow}(\sigma)) \leq v^{\uparrow}(A)$$

Proof. For as long as possible, we do not distinguish the optimistic and pessimistic cases. If the inequality is false, then there is a real $\alpha \in \mathbb{R}$ such that $\min(\kappa(\emptyset), \inf_{i\in I} \sup_{\sigma:A/e_i} v(\sigma)) > \alpha > v(A)$, which in turn implies:

(2) $\kappa(\emptyset) > \alpha$

(3) $\forall i \in I,\ \exists \sigma_i : A/e_i,\ \forall \tau : A^{\perp}/e_i,\ r(\sigma_i, \tau) > \alpha$

(4) $\forall \sigma : A,\ \exists \tau : A^{\perp},\ r(\sigma, \tau) < \alpha$

In particular, (3) gives a family $(\sigma_i)_{i\in I}$. Instantiating (4) to $\mathrm{case}_{i\in I}\sigma_i$, we get $\tau : T \to A^{\perp}$ such that $r(\mathrm{case}_{i\in I}\sigma_i, \tau) < \alpha$.

(5) $\kappa(\emptyset) > \alpha$

(6) $\forall i \in I,\ \forall t,\ \tau(t) = e_i \Rightarrow r(\sigma_i, \tau/t) > \alpha$

(7) $r(\mathrm{case}_{i\in I}, \tau) < \alpha$

Pessimistic case. Since $r(\mathrm{case}_{i\in I}\sigma_i, \tau) < \alpha$, there must be $y \in \langle \mathrm{case}_{i\in I}\sigma_i, \tau\rangle$ such that $\kappa(y) < \alpha$. If T has no minimal $+$-event, then necessarily we have $y = \emptyset$, therefore $\kappa_A(y) = \kappa_A(\emptyset) > \alpha$, contradiction. Therefore, T has a minimal $+$-event. Then by our analysis of interactions for case, there is a minimal $+$-event $t \in T$ and $\tau(t) = e_{i_0}$ and $y' \in \langle \sigma_{i_0}, \tau/t\rangle$ such that $\kappa(y') = \kappa(y) < \alpha$. But this is absurd by (6), so we have found a contradiction.

Optimistic case. By (7) instantiated to the pessimistic case we have that for all $y \in \langle \mathrm{case}_{i\in I}, \tau\rangle$, $\kappa(y) < \alpha$. Take one such $y \in \langle \mathrm{case}_{i\in I}, \tau\rangle$ ($\langle \mathrm{case}_{i\in I}, \tau\rangle$ is non-empty by Zorn's lemma). As above, y cannot be empty as that would cause a contradiction, and T must have a minimal $+$-event. Therefore, there is a minimal $+$-event $t \in T$ and $\tau(t) = e_{i_0}$ and $y' \in \langle \sigma_{i_0}, \tau/t\rangle$ such that $\kappa(y') = \kappa(y) < \alpha$, contradicting (6). □

4.3 Value theorem

Let A be a fixed well-founded and race-free game.

Lemma 4.11 *Let $x \in \mathcal{C}(A)$. Let $(e_i)_{i\in I}$ be the family of extensions of x of negative polarity, and $(f_j)_{j\in J}$ be the family of extensions of x of positive polarity. Then,*

$$v^{\uparrow}(x) = \max(\min(\kappa(x), \inf_{i\in I} v^{\uparrow}(x \cup \{e_i\})), \sup_{j\in J} v^{\uparrow}(x \cup \{f_j\}))$$

$$v^{\downarrow}(x) = \max(\min(\kappa(x), \inf_{i\in I} v^{\downarrow}(x \cup \{e_i\})), \sup_{j\in J} v^{\downarrow}(x \cup \{f_j\}))$$

*Where the **value** $v(x)$ of a configuration $x \in \mathcal{C}(A)$ is defined as $v(A/x)$.*

Proof. The reasoning is the same in the optimistic and pessimistic cases, so we do not distinguish them.

$\leq$. Let $\sigma : S \to A/x$ be a strategy. If there is a minimal event $s \in S$ with $pol(s) = +$, then $v(x) \leq v(\sigma) \leq \sup_{j\in J} v(x \cup \{f_j\})$ by Lemma 4.9. Otherwise, there is no such minimal $s \in S$. Then $v(\sigma) \leq \kappa(x)$. Indeed, letting $\tau : T \to A/x$ be the empty strategy closed by receptivity, we have $\langle \sigma, \tau\rangle = \{\emptyset\}$ and $r(\sigma, \tau) = \kappa(x)$. Similarly taking $i_0 \in I$, by Lemma 4.8 we have $v(\sigma) \leq v(\sigma/e_{i_0})$, and therefore $v(\sigma) \leq \inf_{i\in I} v(x \cup \{e_i\})$.

$\geq$. Let us prove that $\sup_{j\in J} v(x \cup \{f_j\}) \leq v(x)$, taking $j_0 \in J$ and $\sigma : A/(x \cup \{f_{j_0}\})$. Then by Lemma 4.8 we have $v(\mathrm{play}_{f_{j_0}}\sigma) \leq v(\sigma)$ and $v(\sigma) \leq v(x)$. Finally, by Lemma 4.10 we have as needed $\min(\kappa(x), \inf_{i\in I} v(x \cup \{e_i\})) \leq v(x)$. □

Theorem 4.12 *If A is well-founded and race-free then A has a value,* i.e. *we have:*

$$v^{\uparrow}(A) = v^{\downarrow}(A) \qquad v(A) = -v(A^{\perp})$$

*(Note that the second equality only makes sense because by the first, we can talk in a non-ambiguous way of the **value** $v(A)$ of a game A.)*

Proof. Relatively direct consequence of Lemma 4.11. □

We say that a strategy $\sigma : S \to A$ is **optimal** when its *pessimistic value* is equal to the value of the game. Note that it also implies that the optimistic value is equal to the value of the game, since for all $\sigma : S \to A$ we must have $v^{\downarrow}(\sigma) \leq v^{\uparrow}(\sigma) \leq v(A)$.

It also follows that for optimal strategies, the pessimistic and optimistic values coincide. When σ is optimal, we will therefore sometimes just write $v(\sigma)$ for its value.

Example 4.13 Any well-founded race-free game has a value. However this value is not necessarily reached: there are games without optimal strategies. Consider the game A with events $\{\oplus_i \mid i \in \mathbb{N}\}$, pairwise inconsistent, with $\kappa(\emptyset) = 0$ and $\kappa(\{\oplus_i\}) = i$. Its value is $+\infty$ since each positive natural number can be reached, but no strategy σ satisfies $v^{\downarrow}(\sigma) = +\infty$ (though the strategy that plays a nondeterministic choice of natural number satisfies $v^{\uparrow}(\sigma) = +\infty$).

5 Compositionality of optimal strategies

Finally we study how payoff relates to the composition of strategies. We hope that thinking compositionally about values and optimal strategies can be helpful in computing values and optimal strategies for complex games from smaller ones. There are two main kinds of composition of strategies. The first is the categorical composition $\tau \odot \sigma$ of $\sigma : S \to A^{\perp} \parallel B$ and $\tau : T \to B^{\perp} \parallel C$. The second is parallel composition $\sigma \parallel \tau : S \parallel T \to A \parallel B$.

We start this section with the observation that for any strategy $\sigma : S \to A$ we have that $v^{\downarrow}(\sigma) = \inf\{\kappa(\sigma x) \mid x \in \mathcal{C}(S) \ \ +\text{-maximal}\}$, since the definition of pessimistic value quantifies at the same time over Opponent strategies and resulting interactions. From this, we get:

Proposition 5.1 *For strategies $\sigma : S \to A^{\perp} \parallel B$ and $\tau : T \to B^{\perp} \parallel C$, we have $v^{\downarrow}(\tau \odot \sigma) \geq v^{\downarrow}(\tau) + v^{\downarrow}(\sigma)$. Likewise for $\sigma : S \to A$ and $\tau : T \to B$, we have $v^{\downarrow}(\sigma \parallel \tau) = v^{\downarrow}(\sigma) + v^{\downarrow}(\tau)$.*

For categorical composition, $v^{\downarrow}(\tau \odot \sigma) \leq v^{\downarrow}(\tau) + v^{\downarrow}(\sigma)$ does not hold in general, and neither do the two inequalities in the optimistic case. However, the situation is different for *optimal strategies*. To establish this, we first note:

Proposition 5.2 *For race-free, well-founded A and B, $v(A \parallel B) = v(A) + v(B)$.*

Proof. By the value theorem, it does not matter whether we work on the optimistic or pessimistic cases. By simplicity, let us pick the pessimistic one. Firstly, we prove that $v(A \parallel B) \geq v(A) + v(B)$. Indeed, let $\sigma : S \to A$ and $\tau : T \to B$ be strategies. Then, as needed we have $v^{\downarrow}(\sigma \parallel \tau) \geq v^{\downarrow}(\sigma) + v^{\downarrow}(\tau)$ by Proposition 5.1.

Moreover, this inequality also holds for $A^{\perp}$ and $B^{\perp}$, therefore $v(A^{\perp} \parallel B^{\perp}) \geq v(A^{\perp}) + v(B^{\perp})$, from which it follows that $v(A \parallel B) \leq v(A) + v(B)$ by the value theorem and the definition of the dual of games with payoff. □

Theorem 5.3 *If $\sigma : S \to A^{\perp} \parallel B$ and $\tau : T \to B^{\perp} \parallel C$ are optimal strategies, so is $\tau \odot \sigma$. Moreover copycat is optimal, therefore there is a bicategory of concurrent games with payoff and optimal strategies.*

Proof. Suppose σ and τ optimal. We reason as follows:

$$v^{\downarrow}(\tau \odot \sigma) \geq v^{\downarrow}(\sigma) + v^{\downarrow}(\tau) = v(A^{\perp} \parallel B) + v(B^{\perp} \parallel C) = v(A^{\perp} \parallel C)$$

This implies that $v^{\downarrow}(\tau \odot \sigma) = v(A^{\perp} \parallel C)$, since a strict inequality would contradict the definition of $v(A^{\perp} \parallel C)$.

Any $+$-maximal $x \in \mathcal{C}(A^{\perp} \parallel A)$ has the form $y \cup \overline{y}$, where $y \in \mathcal{C}(A)$. Moreover, $\kappa_{A^{\perp}\parallel A}(x) = \kappa_A(y) - \kappa_A(y) = 0$, therefore we have $v^{\downarrow}(\gamma_A) = 0$. However we also have $v(A^{\perp} \parallel A) = v(A) - v(A) = 0$, therefore copycat is optimal. □

We finish this section by remarking that from the theorem above it follows that when σ and τ are optimal, we have $v(\tau \odot \sigma) = v(\sigma) + v(\tau)$, since both sides are forced by optimality to coincide with the value of the game.

6 Conclusion

We have proved a value theorem for race-free well-founded concurrent games. Note that this theorem is not an equilibrium theorem since the value is not always reached. However it is always reached in finite games. In fact our constructions on strategies give an algorithm to compute the value and optimal strategies for finite games. In future we plan to investigate the existence and computation of equilibria in the non-zero-sum case. This will require the extension of our framework to deal with probabilistic strategies, and should allow us to formulate a better connection with the concurrent games of [3,7].

We proved that optimal strategies are stable under composition, forming a bicategory. This compositional structure is worth investigating further. In other work, we have developped an extension of concurrent games *with symmetry*, where events can be duplicated and hence form the basis for a *cartesian closed category* of concurrent strategies. We plan to investigate extensions of the present development in the presence of symmetry, thus providing the basis for a *concurrent programming language* based on the simply-typed λ-calculus and concurrent operations on strategies, for which typable terms describe optimal strategies.

Acknowledgment

The authors gratefully acknowledge the support of the ERC Advanced Grant ECSYM.

References

[1] Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. Full abstraction for pcf. *Inf. Comput.*, 163(2):409–470, 2000.

[2] Samson Abramsky and Paul-André Melliès. Concurrent games and full completeness. In *LICS*, pages 431–442. IEEE Computer Society, 1999.

[3] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002.

[4] Krishnendu Chatterjee and Thomas A. Henzinger. A survey of stochastic ω-regular games. *J. Comput. Syst. Sci.*, 78(2):394–413, 2012.

[5] Pierre Clairambault, Julian Gutierrez, and Glynn Winskel. The winning ways of concurrent games. In *LICS*. IEEE Computer Society, 2012.

[6] John Conway. *On Numbers and Games.* Wellesley, MA: A K Peters, 2000.

[7] Luca de Alfaro and Thomas A. Henzinger. Concurrent omega-regular games. In *LICS*, pages 141–154. IEEE Computer Society, 2000.

[8] J. M. E. Hyland and C.-H. Luke Ong. On full abstraction for pcf: I, ii, and iii. *Inf. Comput.*, 163(2):285–408, 2000.

[9] Martin Hyland. Game semantics. In *Semantics and logics of computation*, Publications of the Newton Institute. Cambridge University Press, 1997.

[10] Andre Joyal. Remarques sur la théorie des jeux à deux personnes. *Gazette des sciences mathématiques du Québec, 1(4)*, 1977.

[11] Joachim Lambek and Philip J. Scott. *Introduction to higher order categorical logic*. Cambridge Univiversity Press, 1988.

[12] Donald A. Martin. Borel determinacy. *The Annals of Mathematics*, 102(2):363–371, 1975.

[13] Donald A. Martin. The determinacy of blackwell games. *J. Symb. Log.*, 63(4):1565–1581, 1998.

[14] Paul-André Melliès. Asynchronous games 4: A fully complete model of propositional linear logic. In *LICS*, pages 386–395. IEEE Computer Society, 2005.

[15] Paul-André Melliès and Samuel Mimram. Asynchronous games: Innocence without alternation. In Luís Caires and Vasco Thudichum Vasconcelos, editors, *CONCUR*, volume 4703 of *Lecture Notes in Computer Science*, pages 395–411. Springer, 2007.

[16] Mogens Nielsen, Gordon D. Plotkin, and Glynn Winskel. Petri nets, event structures and domains. In *Semantics of Concurrent Computation*, volume 70 of *Lecture Notes in Computer Science*. Springer, 1979.

[17] Silvain Rideau and Glynn Winskel. Concurrent strategies. In *LICS*, pages 409–418. IEEE Computer Society, 2011.

[18] Glynn Winskel. Event structures. In Wilfried Brauer, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Advances in Petri Nets*, volume 255 of *Lecture Notes in Computer Science*, pages 325–392. Springer, 1986.

A Preliminaries: stable families and composition

The detailed proofs often rely on stable families. Constructions such as product and pullback are often done most conveniently in the category of stable families. There is a full and faithful embedding of the category of event structures into the category of stable families. It has a right adjoint which translates limits such as pullback and product of stable families to the corresponding universal constructions in the category of event structures. [3]

Definition A.1 For $\sigma : S \to A$ and $\tau : T \to A^\perp$ we will write $[\sigma, \tau]$ for the set of **interactions** between σ and τ, *i.e.* maximal configurations of the stable family $\mathcal{C}(T) \odot \mathcal{C}(S)$. (It is the pullback of σ and τ in the category of stable families.)

B Constructions on strategies

Lemma B.1 *Let $\sigma : S \to A$ be a strategy. Then $\sigma \cong \sigma'$, a strategy $\sigma' : S' \to A$ for which*

$$\forall s' \in S'.\ pol_{S'}[s']_{S'} = \{-\} \Rightarrow \sigma'(s') = s'\,. \qquad (\dagger)$$

Moreover,

$$S'_- = A_-$$

where for an event structure with polarity E we write $E_- =_{\mathrm{def}} \{e \in E \mid pol[e] \subseteq \{-\}\}$.

[3] A recent reference: Glynn Winskel. Event structures, stable families and games. Lecture notes, Comp Science Dept, Aarhus University, Available from http://daimi.au.dk/~gwinskel, 2011.

Proof. As a consequence of receptivity and negative innocence [LICS11], whenever $\emptyset \subseteq^- y$ in $\mathcal{C}(A)$ there is a *unique* $x \in \mathcal{C}(S)$ so that $\emptyset \subseteq^- x$ & $\sigma x = y$. Consequently, the map σ induces an order isomorphism w.r.t. inclusion between configurations $x \in \mathcal{C}(S)$ where $\emptyset \subseteq^- x$ and $y \in \mathcal{C}(A)$ where $\emptyset \subseteq^- y$. The order isomorphism restricts to an order isomorphism between prime configurations. It follows that σ is bijective between events $s \in S_-$ and events $a \in A_-$. This bijection extends to a bijective renaming of events of S to those of S'. □

The lemma permits us to assume strategies satisfy (†) in the following results.

Proposition B.2 *Let $\sigma_i : S_i \to A$, for $i \in I$, be strategies (assumed to satisfy (†)). W.l.og. we may assume that whenever indices $i, j \in I$ are distinct then so are those events of S_i and S_j which causally depend on a positive event (otherwise we could tag such events by their respective indices). Define S to be the event structure with events $\bigcup_{i\in I} S_i$, causal dependency $s \leq_S e'$ iff $s \leq_{S_i} e'$, for some $i \in I$, and consistency $X \in \mathrm{Con}_S$ iff $X \in \mathrm{Con}_{S_i}$, for some $i \in I$. Defining $\|_{i\in I}\, \sigma_i(s) = \sigma_i(s)$ if $s \in S_i$ yields a strategy $\|_{i\in I}\, \sigma_i : S \to A$.*

Proof. *Pre-strategy.* Follows from the observation that for any $x \in \mathcal{C}(\bigcup_{i\in I} S_i)$ there is $i \in I$ such that $x \in \mathcal{C}(S_i)$. Therefore preservation of configuration and local injectivity directly follow from those properties for the σ_is.

Receptivity. Trivial, since (†) is preserved by union and implies receptivity.

Innocence. For any $s_1, s_2 \in \bigcup_{i\in I} S_i$, if $s_1 \rightarrowtriangle s_2$ then there is $i \in I$ such that $s_1, s_2 \in S_i$ and $s_1 \rightarrowtriangle s_2$ in S_i as well. Therefore if $pol(s_1) \neq -$ or $pol(s_2) \neq +$ then by innocence of σ_i we have $\sigma_i(s_1) \rightarrowtriangle \sigma_i(s_2)$ as well, therefore $(\|_{i\in I}\, \sigma_i)(s_1) \rightarrowtriangle (\|_{i\in I}\, \sigma_i)(s_2)$ and $\|_{i\in I}\, \sigma_i$ is innocent. □

Proposition B.3 *Suppose A is a race-free game such that $\emptyset \overset{a}{\longrightarrow\!\!\!\subset}$ with $pol(a) = +$. Then for any strategy $\sigma : S \to A/a$, where w.l.o.g. $a \notin S$, there is a strategy $\mathrm{play}_a(\sigma) : S' \to A$: the event structure S' comprises*

- *events, $S \cup \{a\}$,*
- *causal dependency, that on S extended by $a \leq_{S'} s$, for $s \in S$, whenever $a \leq_A \sigma(s)$,*
- *with consistency, $X \in \mathrm{Con}_{S'}$ iff $X \cap S \in \mathrm{Con}_S$,*

and $\mathrm{play}_a(\sigma)(s) = \sigma(s)$, for $s \in S$, with $\mathrm{play}_a(\sigma)(a) = a$.

Proof. It is easy to check that S' is an event structure and that $\mathrm{play}_a(\sigma)$ is a total map of event structures which preserves polarity. Innocence is inherited from σ. That it is receptive follows from the race-freedom of A: Let $x \in \mathcal{C}(S')$ and $\mathrm{play}_a(\sigma)x \overset{a'}{\longrightarrow\!\!\!\subset}$ where $a' \in A$ and $pol(a') = -$. If $a \in x$ then receptivity condition for $\mathrm{play}_a(\sigma)$ follows directly from that of σ. If $a \notin x$ then $x \in \mathcal{C}(S)$ and $\mathrm{play}_a(\sigma)x = \sigma x$. From the race-fredom of A we deduce that $\sigma x \overset{a'}{\longrightarrow\!\!\!\subset}$ in A/a. Again the receptivity condition for $\mathrm{play}_a(\sigma)$ follows from that of σ. □

Proposition B.4 *Suppose A is a game such that $\emptyset \overset{a}{\longrightarrow\!\!\!\subset}$ with $pol(a) = -$. Then for any strategy $\sigma : S \to A/a$, where w.l.o.g. $a \notin S$, there is a strategy $\mathrm{wait}_a(\sigma) : S' \to A$: the event structure S' comprises*

- *events, $S \cup A_-$, where $A_- =_{\mathrm{def}} \{a' \in A \mid pol_A[a']_A \subseteq \{-\}\}$,*

- *causal dependency, that on S and A_- extended by $a \leq_{S'} s$, for $s \in S$, whenever $a \leq_A \sigma(s)$ or $pol(s) = +$,*
- *with consistency, $X \in \mathrm{Con}_{S'}$ iff $X \cap S \in \mathrm{Con}_S$ & $\mathrm{wait}_a(\sigma)X \in \mathrm{Con}_A$,*

where $\mathrm{wait}_a(\sigma)(s')$ is defined to be $\sigma(s')$ if $s' \in S$, otherwise s'.

Proof. By innocence, the causal dependencies on S and A_- agree where they overlap. As $a \notin S$, by assumption, we obtain a partial order $\leq_{S'}$ from the definition above. It is routine to check that S' is an event structure.

Observe that if $\sigma(s) \in A_-$ then $s \in S_-$, for all $s \in S$: otherwise there would be a maximal positive event on which s causally depended, contradicting $-$-innocence of σ.

In checking that $\mathrm{wait}_a(\sigma)$, clearly a total function, is a map of event structures it is straightforward to show that the image of a configuration $x \in \mathcal{C}(S')$ is down-closed in A. By definition $\mathrm{wait}_a(\sigma)$ preserves consistency, so $\mathrm{wait}_a(\sigma)x$ is also consistent and in $\mathcal{C}(A)$. Suppose now $s_1, s_2 \in x$ with $\mathrm{wait}_a(\sigma)(s_1) = \mathrm{wait}_a(\sigma)(s_2)$. If both $s_1, s_2 \in S$ then $\sigma(s_1) = \sigma(s_2)$ so $s_1 = s_2$ as σ is map of event structures. Otherwise, either $s_1 \notin S$ or $s_2 \notin S$. If both $s_1 \notin S$ and $s_2 \notin S$, then $s_1 = s_2$, directly from the definition of $\mathrm{wait}_a(\sigma)$. Otherwise, w.lo.g. suppose $s_1 \in S$ and $s_2 \notin S$. Then $\sigma(s_1) = s_2$ and $s_2 \in A_-$. By the observation above, $s_1 \in S_-$. But σ is assumed to satisfy (†), so $\sigma(s_1) = s_1 = s_2$. The function $\mathrm{wait}_a(\sigma)$ is indeed a map of event structures.

The map $\mathrm{wait}_a(\sigma)$ clearly preserves polarity. The construction preserves the innocence inherited from σ. We show receptivity. Suppose $x \in \mathcal{C}(S)$ and $\mathrm{wait}_a(\sigma)x \overset{a'}{\longrightarrow\!\!\!\subset}$ in A where a' has negative polarity. Consider first the case when $a' \in A_-$. Then it can be checked that $x \cup \{a'\} \in \mathcal{C}(S')$. This yields $x \overset{a'}{\longrightarrow\!\!\!\subset}$ with $\mathrm{wait}_a(\sigma)(a') = a'$. To show uniquesness, assume $\mathrm{wait}_a(\sigma)(s') = a'$. If $s' \notin S$ we obtain $\mathrm{wait}_a(\sigma)(s') = s' = a'$. If on the other hand, $s' \in S$ we obtain $\mathrm{wait}_a(\sigma)(s') = \sigma(s') = a' \in A_-$. By the observation, $s' \in S_-$ and $\sigma(s') = s'$ as σ satisfies (†), and again $s' = a'$.

In the case where $a' \notin A_-$ there must be $a_1 \leq_A a'$ with $pol(a_1) = +$. Hence there is $s_1 \in x$, with $pol(s_1) = +$, such that $\sigma(s_1) = a_1$. From the causal dependency of S' we must have $a \in x$. It follows that $x \setminus \{a\} \in \mathcal{C}(S)$ and $\sigma(x \setminus \{a\}) \overset{a'}{\longrightarrow\!\!\!\subset}$ in A/a, whereupon receptivity of σ ensures the required receptivity condition for $\mathrm{wait}_a(\sigma)$.□

Proposition B.5 *For $\sigma : S \to A$ a strategy and $x \in \mathcal{C}^\infty(S)$, the function $\sigma/s : S/s \to A/\sigma(s)$ is a strategy.*

Proof. A straightforward check. □

Lemma B.6 *For all $\sigma : S \to A$ and $\tau : T \to A^\perp$, then*

$$\kappa\langle\sigma, \tau\rangle = \{-v \mid v \in \kappa\langle\tau, \sigma\rangle\}$$

Proof. Straightforward. □

For all the forthcoming lemmas, the well-founded hypothesis is not strictly necessary. However we keep it as it simplifies the proofs, and these lemmas will only be applied on well-founded games in order to get the theorem.

Lemma B.7 *If the game A is well-founded and race-free,*

$$\begin{aligned}\kappa\langle \mathop{\|}_{i\in I} \sigma_i, \tau\rangle &\subseteq \bigcup_{i\in I} \kappa\langle \sigma_i, \tau\rangle\\ \kappa\langle \mathop{\|}_{i\in I} \sigma_i, \tau\rangle^+ &= \bigcup_{i\in I} \kappa\langle \sigma_i, \tau\rangle^+\end{aligned}$$

Proof. First, we prove that $\kappa\langle \|_{i\in I} \sigma_i, \tau\rangle \subseteq \bigcup_{i\in I} \kappa\langle\sigma_i, \tau\rangle$. Take $y \in \langle \|_{i\in I} \sigma_i, \tau\rangle$. Necessarily, there is $z \in [\|_{i\in I} \sigma_i, \tau]$ such that $\sigma\Pi_1 z = y$. By definition of $\|_{i\in I} \sigma_i$, there is $i \in I$ such that $\Pi_1 z \in S_i$. It follows that $z \in [\sigma_i, \tau]$, therefore $y \in \langle \sigma_i, \tau\rangle$ as well.

Likewise if $y \in \langle\sigma_i, \tau\rangle$ with a positive event, take its witness $z \in [\sigma_i, \tau]$. Obviously $z \in \mathcal{C}(T) \odot \mathcal{C}(S')$ (where $\|_{i\in I} \sigma_i : S' \to A$). Maximality follows from that of z in $\mathcal{C}(T) \odot \mathcal{C}(S_i)$: indeed since y has a $+$-event this event is only consistent with events in S_i, hence any extension of z must be compatible with S_i. □

Lemma B.8 *If A is race-free and well-founded, then,*

$$\kappa\langle \mathrm{play}_a(\sigma), \tau\rangle = \kappa\langle\sigma, \tau/a\rangle$$

Proof. First we prove that $\kappa\langle \mathrm{play}_a(\sigma), \tau\rangle \subseteq \kappa\langle\sigma, \tau/a\rangle$. Take $y \in \langle \mathrm{play}_a(\sigma), \tau\rangle$ and its witness $z \in [\mathrm{play}_a(\sigma), \tau]$ such that $y = \mathrm{play}_a(\sigma)z$. The difficult part of the proof consist in proving that $a \in y$, let us start with that. Suppose that $a \notin y$. Obviously if $y \cup \{a\} \in \mathrm{Con}_A$, we have a contradiction with the maximality of z. Otherwise if $a \notin y$ but $y \cup \{a\} \notin \mathrm{Con}_A$, then consider a subconfiguration $y' \subseteq y$ that is minimal such that $y' \cup \{a\} \notin \mathrm{Con}_A$, that is, all the subconfigurations of y' are compatible with a. Necessarily y' is non-empty, otherwise it would be compatible with a, take $y'' {-\!\!\subset} y'$, write e the event such that $y'' \cup \{e\} = y'$. If $pol(e) = -$ then by race-freedom of A we have $y' \cup \{a\} \in \mathrm{Con}_A$ as well, contradiction, therefore $pol(e) = +$. Consider the witnesses $z', z'' \in [\mathrm{play}_a(\sigma), \tau]$ corresponding to y', y'', and $u' = \Pi_1 z'$, $u'' = \Pi_1 z''$, with $u' {-\!\!\subset} u''$ and $u' \cup \{s\} = u''$, with $\mathrm{play}_a(\sigma)(s) = e$. Then take $u'' \cap S$, it is still a configuration of S' (with $\mathrm{play}_a(\sigma) : S' \to A$). Necessarily we have $(\mathrm{play}_a(\sigma)(u'' \cap S)) \cup \{a\} \in \mathcal{C}(A)$ since σ is a strategy on A/a, and we also have $u'' \cap S \subseteq^- u''$ where $(\mathrm{play}_a(\sigma)u'') \cup \{a\} \notin \mathcal{C}(A)$, but this is forbidden by race-freedom of A, contradiction.

Therefore, $a \in y$. Then we have $(a, \overline{a}) \in z$. Set $z' = z \setminus \{(a, \overline{a})\}$, it is straightforward to check that $z' \in [\sigma, \tau/a]$ and $\sigma\Pi_1 z' = y \setminus \{a\}$, therefore $\kappa(\sigma\Pi_1 z') = \kappa(y)$ by definition of κ on A/a.

We now turn to the other inequality. Take $y \in \kappa\langle\sigma, \tau/a\rangle$ along with its witness $z \in [\sigma, \tau/a]$. Then it is straightforward to check that $z' = z \cup \{(a, \overline{a})\} \in [\mathrm{play}_a(\sigma), \tau]$ and $\kappa(\mathrm{play}_a(\sigma)z') = \kappa(y)$ by definition of κ on A/a. □

Lemma B.9 *We have the following* equalities *between strategies:*

$$\begin{aligned}&\mathrm{play}_a(\sigma)/a = \sigma\\ &\mathrm{wait}_a(\sigma)/a = \sigma\end{aligned}$$

Proof. Trivial. □

Lemma B.10 *If A is well-founded and race-free, then,*

$$\kappa\langle \mathrm{play}_a(\sigma), \mathrm{wait}_a(\tau)\rangle = \kappa\langle\sigma,\tau\rangle$$

Proof. Trivial using Lemmas B.8 and B.9. □

Lemma B.11 *If A is well-founded and race-free, then,*

$$\begin{aligned}\kappa\langle \mathrm{wait}_a(\sigma), \tau\rangle &\supseteq \bigcup_{t:\tau(t)=a} \kappa\langle\sigma,\tau/t\rangle\\ \kappa\langle \mathrm{wait}_a(\sigma), \tau\rangle^+ &= \bigcup_{t:\tau(t)=a} \kappa\langle\sigma,\tau/t\rangle^+\end{aligned}$$

Proof. We start with the left-to-right inclusion, take $y \in \langle \mathrm{wait}_a(\sigma), \tau\rangle$ (supposed to have positive events) along with its witness $z \in [\mathrm{wait}_a(\sigma), \tau]$. Since y has positive events it must contain a, as positive events in $\mathrm{wait}_a(\sigma) : S' \to A$ are set to depend on a. Therefore there is some $t \in T$ such that $\tau(t) = a$ and $(a, t) \in z$. Defining $z' = z \setminus (a, t)$, it is straightforward to prove that $z' \in [\sigma, \tau/t]$, and $\kappa(\sigma\pi_1 z) = \kappa(y)$ by definition of κ on A/a.

Reciprocally take $t \in T$ such that $\tau(t) = a$, and $y \in \langle\sigma, \tau/t\rangle$ with its witness $z \in [\sigma, \tau/t]$. Then it is straightforward to prove that $z' = z \cup (a, t) \in [\mathrm{wait}_a(\sigma), \tau]$, and $\kappa((\mathrm{wait}_a(\sigma))\pi_1 z') = \kappa(y)$ by definition of κ on A/a. Take $x \in \mathcal{C}(T)$ +-maximal and such that $pol\ x \subseteq \{+\}$ with $a \notin x$, then define $z = \{(\bar{e}, e) \mid e \in x\}$. Then it is straightforward to check that $z \in \mathcal{C}(T) \odot \mathcal{C}(S')$, and z is maximal: indeed $\pi_2 z = y$ is +-maximal, and $\pi_1 z$ is +-maximal as well by definition of $\mathrm{wait}_a(\sigma)$ since $a \notin x$. It follows that $z \in [\mathrm{wait}_a(\sigma), \tau]$ with as required $\kappa(\mathrm{wait}_a(\sigma))\pi_1 z = \kappa\overline{\tau x}$. □

Corollary B.12 *Setting* $\mathrm{case}_{i\in I}\sigma_i = \parallel_{i\in I} \mathrm{wait}_{a_i}(\sigma_i)$*, and if $\tau : T \to A^\perp$ is such that T has a minimal +-event, then.*

$$\begin{aligned}\kappa\langle \mathrm{case}_{i\in I}\sigma_i, \tau\rangle &\subseteq \bigcup_{i\in I}\ \bigcup_{t:\tau(t)=a_i} \kappa\langle\sigma_i,\tau/t\rangle\\ \kappa\langle \mathrm{case}_{i\in I}\sigma_i, \tau\rangle^+ &= \bigcup_{i\in I}\ \bigcup_{t:\tau(t)=a_i} \kappa\langle\sigma_i,\tau/t\rangle^+\end{aligned}$$

If T has no +-minimal event, then $\kappa\langle \mathrm{case}_{i\in I}\sigma_i, \tau\rangle = \{\kappa(\emptyset)\}$.

Proof. We apply the following reasoning, putting all the previous lemmas together:

$$\begin{aligned}\kappa\langle \mathrm{case}_{i\in I}\sigma_i, \tau\rangle &= \kappa\langle \parallel_{i\in I} \mathrm{wait}_{a_i}(\sigma_i), \tau\rangle\\ &\subseteq \bigcup_{i\in I} \kappa\langle \mathrm{wait}_{a_i}(\sigma_i), \tau\rangle\\ &\subseteq \bigcup_{i\in I}(\ \bigcup_{t:\tau(t)=a_i} \kappa\langle\sigma_i,\tau/t\rangle\end{aligned}$$

All these inclusions become equalities when restricted to configurations with a positive event. □

C Results of these constructions

Lemma C.1 *For any well-founded race-free game A and $a \in A$ with $pol(a) = +$ such that $\emptyset \overset{a}{\longrightarrow\!\!\!\subset}$, for any strategy $\sigma : S \to A/a$, we have:*

$$v^{\uparrow}(\mathrm{play}_a(\sigma)) \leq v^{\uparrow}(\sigma)$$
$$v^{\downarrow}(\mathrm{play}_a(\sigma)) \leq v^{\downarrow}(\sigma)$$

Proof. First inequality:

$$v^{\uparrow}(\mathrm{play}_a(\sigma) \leq v^{\uparrow}(\sigma)$$

Let $\tau : T \to A^{\perp}/a$, and $z \in \langle \mathrm{play}_a\sigma, \mathrm{wait}_a\tau\rangle$. By Lemma B.10, there is $z' \in \langle \sigma, \tau\rangle$ such that $\kappa(z) = \kappa(z')$.

Second inequality:

$$v^{\downarrow}(\mathrm{play}_a\sigma) \leq v^{\downarrow}(\sigma)$$

Let $\tau : T \to A^{\perp}/a$ and $z \in \langle \sigma, \tau\rangle$. Then by Lemma B.10 there is $z' \in \langle \mathrm{play}_a\sigma, \mathrm{wait}_a\tau\rangle$ such that $\kappa(z) = \kappa(z')$. □

Lemma C.2 *For any well-founded race-free game A, $a \in A$ with $pol(a) = -$ such that $x \overset{a}{\longrightarrow\!\!\!\subset}$, for all strategy $\sigma : S \to A/x$, we have:*

$$v^{\uparrow}(\sigma) \leq v^{\uparrow}(\sigma/a)$$
$$v^{\downarrow}(\sigma) \leq v^{\downarrow}(\sigma/a)$$

Proof. First inequality:

$$v^{\uparrow}(\sigma) \leq v^{\uparrow}(\sigma/a)$$

Let $\tau : T \to A^{\perp}/(x \cup \{a\})$, and $z \in \langle \sigma, \mathrm{play}_a\tau\rangle$. By Lemma B.8 there is $z' \in \langle \sigma/a, \tau\rangle$ such that $\kappa(z) = \kappa(z')$.

Second inequality:

$$v^{\downarrow}(\sigma) \leq v^{\downarrow}(\sigma/a)$$

Let $\tau : T \to A^{\perp}/(x \cup \{a\})$, and $z \in \langle \sigma/a, \tau\rangle$. Then by Lemma B.8 there is $z' \in \langle \sigma, \mathrm{play}_a\tau\rangle$ such that $\kappa(z) = \kappa(z')$. □

Lemma C.3 *Suppose A is race-free, $x \in \mathcal{C}^{\infty}(A)$. Let $(f_j)_{j \in J}$ be the family of minimal $+$-events of A. Let $\sigma : S \to A$ be a strategy such that there is a minimal $+$-event $s \in S$. Then,*

$$v^{\downarrow}(\sigma) \leq \sup_{j \in J} v^{\downarrow}(\{f_j\})$$
$$v^{\uparrow}(\sigma) \leq \sup_{j \in J} v^{\uparrow}(\{f_j\})$$

Proof. *Pessimistic case.* Necessarily there must be $j_0 \in J$ such that $\sigma(s) = f_{j_0}$. Then, we are going to prove that

$$v^{\downarrow}(\sigma) \leq v^{\downarrow}(\sigma/s)$$

Indeed, take $\tau : A^{\perp}/f_{j_0}$, and $z \in \langle \sigma/s, \tau \rangle$. By Lemma B.11, there is $z' \in \langle \sigma, \mathrm{wait}_{f_{j_0}}(\tau) \rangle$ such that $\kappa(z) = \kappa(z')$.

Optimistic case. Suppose that the inequality is false, *i.e.*

$$\sup_{j \in J} v^{\uparrow}(\{f_j\}) < v^{\uparrow}(\sigma)$$

This implies that there is $\alpha \in \mathbb{R}$ such that $\sup_{j \in J} v^{\uparrow}(\{f_j\}) < \alpha$ and $v^{\uparrow}(\sigma) > \alpha$. The first inequality implies:

$$\forall j \in J,\ \forall \sigma : A'/f_j,\ \exists \tau' : A^{\perp}/f_j,\ \forall z' \in \langle \sigma', \tau' \rangle,\ \kappa(z') < \alpha$$

which is easily shown to imply:

$$\forall (\sigma_k)_{k \in K},\ \exists (\tau_j)_{j \in \sigma K},\ \forall k \in K,\ \forall z' \in \langle \sigma_k, \tau_{\sigma k} \rangle,\ \kappa(z') < \alpha \tag{C.1}$$

where K is the set of positive minimal events in S. Applying this property to the family of strategies obtained by $\sigma_k = \sigma/k$, we get a family of counter-strategies $(\tau_j)_{j \in \sigma K}$. We extend this family to J by setting τ_j as the empty strategy (closed under receptivity) whenever $e_j \notin \sigma K$. Thus, we get a family $(\tau_j)_{j \in J}$.

Likewise, the second inequality implies that:

$$\forall \tau : A^{\perp},\ \exists z \in \langle \sigma, \tau \rangle,\ \kappa(z) > \alpha$$

Let us apply it to $\tau = \mathrm{case}_{j \in J} \tau_j$, we get $z \in \langle \sigma, \mathrm{case}_{j \in J} \tau_j \rangle$ such that $\kappa(z) > \alpha$. By Corollary B.12, there is $k_0 \in K$, and $z' \in \langle \sigma/k_0, \tau_{\sigma k_0} \rangle$ such that $\kappa(z') = \kappa(z) > \alpha$. However, applying (1) to k_0 also shows that $\kappa(z') < \alpha$, contradiction. Hence, the required inequality is true. □

Lemma C.4 *Let A be a game, $(e_i)_{i \in I}$ the family of its negative minimal events. Then,*

$$\min(\kappa(\emptyset), \inf_{i \in I} \sup_{\sigma : A/e_i} v^{\downarrow}(\sigma)) \leq v^{\downarrow}(A)$$

$$\min(\kappa(\emptyset), \inf_{i \in I} \sup_{\sigma : A/e_i} v^{\uparrow}(\sigma)) \leq v^{\uparrow}(A)$$

Proof. For as long as possible, we do not distinguish the optimistic and pessimistic cases. Suppose that the inequality is false. It implies that there is $\alpha \in \mathbb{R}$ such that

$$\min(\kappa(\emptyset), \inf_{i \in I} \sup_{\sigma : A/e_i} v(\sigma)) > \alpha$$

$$v(A) < \alpha$$

which imply the following three propositions:

$$\kappa(\emptyset) > \alpha \tag{C.2}$$

$$\forall i \in I,\ \exists \sigma_i : A/e_i,\ \forall \tau : A^{\perp}/e_i,\ r(\sigma_i, \tau) > \alpha \tag{C.3}$$

$$\forall \sigma : A,\ \exists \tau : A^{\perp},\ r(\sigma, \tau) < \alpha \tag{C.4}$$

In particular, (C.3) gives a family $(\sigma_i)_{i \in I}$. Instanciating (C.4) with $\mathrm{case}_{i \in I} \sigma_i$, we get $\tau : T \to A^{\perp}$ such that $r(\mathrm{case}_{i \in I} \sigma_i, \tau) < \alpha$.

$$\kappa(\emptyset) > \alpha \tag{C.5}$$

$$\forall i \in I,\ \forall t,\ \tau(t) = e_i \Rightarrow r(\sigma_i, \tau/t) > \alpha \tag{C.6}$$

$$r(\mathrm{case}_{i \in I}, \tau) < \alpha \tag{C.7}$$

Let us now distinguish the optimistic and pessimistic cases.

Pessimistic case. Since $r(\text{case}_{i\in I}\sigma_i, \tau) < \alpha$, there must be $y \in \langle \text{case}_{i\in I}\sigma_i, \tau\rangle$ such that $\kappa(y) < \alpha$. If T has no minimal +-event, then necessarily we have $y = \emptyset$, therefore $\kappa_A(y) = \kappa_A(\emptyset) > \alpha$, contradiction. Therefore, T has a minimal +-event. Then by Corollary B.12 there is a minimal +-event $t \in T$ and $\tau(t) = e_{i_0}$ and $y' \in \langle \sigma_{i_0}, \tau/t\rangle$ such that $\kappa(y') = \kappa(y) < \alpha$. But this is absurd by (C.6), so we have found a contradiction.

Optimistic case. By (C.7) instanciated in the pessimistic case we have that for all $y \in \langle \text{case}_{i\in I}, \tau\rangle$, $\kappa(y) < \alpha$. Take one such $y \in \langle \text{case}_{i\in I}, \tau\rangle$ ($\langle \text{case}_{i\in I}, \tau\rangle$ is non-empty by Zorn's lemma). As above, y cannot be empty as that would cause a contradiction, and T must have a minimal +-event. Therefore by Corollary B.12 there is a minimal +-event $t \in T$ and $\tau(t) = e_{i_0}$ and $y' \in \langle \sigma_{i_0}, \tau/t\rangle$ such that $\kappa(y') = \kappa(y) < \alpha$, contradicting (C.6). □

D Proof of the value theorem

Let A be a fixed well-founded and race-free game.

Lemma D.1 *Let $x \in \mathcal{C}(A)$. Let $(e_i)_{i\in I}$ be the family of extensions of x of negative polarity, and $(f_j)_{j\in J}$ be the family of extensions of x of positive polarity. Then,*

$$v^\uparrow(x) = \max(\min(\kappa(x), \inf_{i\in I} v^\uparrow(x \cup \{e_i\})), \sup_{j\in J} v^\uparrow(x \cup \{f_j\}))$$
$$v^\downarrow(x) = \max(\min(\kappa(x), \inf_{i\in I} v^\downarrow(x \cup \{e_i\})), \sup_{j\in J} v^\downarrow(x \cup \{f_j\}))$$

Proof. The reasoning is the same in the optimistic and pessimistic cases, hence we do not distinguish them. We prove the first unequality:

$$v(x) \leq \max(\min(\kappa(x), \inf_{i\in I} v(x \cup \{e_i\})), \sup_{j\in J} v(x \cup \{f_j\}))$$

Let $\sigma : S \to A/x$ be a strategy. If there is a minimal event $s \in S$ with $pol(s) = +$, then $v(x) \leq v(\sigma) \leq \sup_{j\in J} v(x \cup \{f_j\})$ by Lemma C.3. Otherwise, there is no such minimal $s \in S$. Then $v(\sigma) \leq \kappa(x)$. Indeed, let $\tau : T \to A/x$ be the empty strategy closed by receptivity, we have $\langle \sigma, \tau\rangle = \{\emptyset\}$ and $r(\sigma, \tau) = \kappa(x)$. Likewise take $i_0 \in I$, by Lemma C.2 we have $v(\sigma) \leq v(\sigma/e_{i_0})$, therefore $v(\sigma) \leq \inf_{i\in I} v(x \cup \{e_i\})$.

We now prove the other inequality:

$$\max(\min(\kappa(x), \inf_{i\in I} v(x \cup \{e_i\})), \sup_{j\in J} v(x \cup \{f_j\})) \leq v(x)$$

Let us prove that $\sup_{j\in J} v(x \cup \{f_j\}) \leq v(x)$, taking $j_0 \in J$ and $\sigma : A/(x \cup \{f_{j_0}\})$. Then by Lemma C.1 we have $v(\text{play}_{f_{j_0}}\sigma) \leq v(\sigma)$ and $v(\sigma) \leq v(x)$. Finally, we need to prove that $\min(\kappa(x), \inf_{i\in I} v(x \cup \{e_i\})) \leq v(x)$, but this is Lemma C.4. □

Theorem D.2 *If A is well-founded and race-free, then the optimistic and pessimistic values coincide:*

$$v^\uparrow(A) = v^\downarrow(A)$$

This justifies writing $v(A)$ for the value *of a game.*

Proof. It is obvious from the lemma above that there cannot be a maximal $x \in \mathcal{C}(A)$ maximal such that $v^\uparrow(A/x) \neq v^\downarrow(A/x)$. Since A is well-founded, that must be true for the empty configuration. □

Theorem D.3 *If A is well-founded and race-free, then we have:*

$$v(A) = -v(A^\perp)$$

Proof. Let $x \in \mathcal{C}(A)$ be maximal such that $v(A/x) = -v(A^\perp/x)$. Let $(e_i)_{i\in I}$ be the family of negative extensions of x and $(f_j)_{j\in J}$ its family of positive extensions. Then,

$$\begin{aligned}
v(A/x) &= \max(\min(\kappa_A(x), \inf_{i\in I} v(A/(x\cup\{e_i\}))), \sup_{j\in J} v(A/(x\cup\{f_j\})))\\
&= \max(\min(-\kappa_{A^\perp}(x), \inf_{i\in I} -v(A^\perp/(x\cup\{e_i\}))), \sup_{j\in J} -v(A^\perp/(x\cup\{f_j\})))\\
&= -\min(\max(\kappa_{A^\perp}(x), \sup_{i\in I} v(A^\perp/(x\cup\{e_i\}))), \inf_{j\in J} v(A^\perp/(x\cup\{f_j\})))\\
&= -\max(\min(\kappa_{A^\perp}(x), \inf_{j\in J} v(A^\perp/(x\cup\{f_j\}))),\\
&\qquad \min(\sup_{i\in I} v(A^\perp/(x\cup\{e_i\})), \inf_{j\in J} v(A^\perp/(x\cup\{f_j\}))))
\end{aligned}$$

But for all $i_0 \in I$, $v(A^\perp/(x\cup\{e_{i_0}\})) \leq v(A^\perp)$ by Lemma C.1 and for all $j_0 \in J$, we have $v(A^\perp) \leq v(A^\perp/(x\cup\{f_{j_0}\}))$ by Lemma C.2, therefore $\sup_{i\in I} v(A^\perp/(x\cup\{e_i\})) \leq \inf_{j\in J} v(A^\perp/(x\cup\{f_j\}))$, and:

$$\begin{aligned}
v(A/x) &= -\max(\min(\kappa_{A^\perp}(x), \inf_{j\in J} v(A^\perp/(x\cup\{f_j\}))), \inf_{j\in J} v(A^\perp/(x\cup\{f_j\})))\\
&= -v(A^\perp/x)
\end{aligned}$$

Contradiction. Therefore there is no such maximal x and the property is true for the empty configuration, thus $v(A) = -v(A^\perp)$ since A is well-founded. □

E Proofs on compositionality of optimal strategies

Proposition E.1 *Let A be a game and $\sigma : S \to A$ a strategy. Then,*

$$v^\downarrow(\sigma) = \inf\{\kappa(\sigma x) \mid x \in \mathcal{C}(S) \quad +\text{-}maximal\}$$

Proof. $\leq$. It suffices to show:

$$\forall x \in \mathcal{C}(S) \quad +\text{-maximal},\ \exists \tau : T \to A^\perp,\ \exists y \in \langle \sigma, \tau\rangle,\ \kappa(y) \leq \kappa(\sigma x)$$

Thus, let $x \in \mathcal{C}(S)$ be $+$-maximal. Set $T = (\sigma x)^\perp$ with $\tau : T \to A^\perp$ acting as the identity on events. τ is obviously innocent but not necessarily receptive, consider its closure $\tau' : T' \to A^\perp$ by receptivity. Then, define:

$$z = \{(e, \overline{\sigma e}) \mid e \in x\}$$

It is straightforward to check that $z \in \mathcal{C}(S) \odot \mathcal{C}(T')$, and it is maximal since x is $+$-maximal and by construction of τ'. It follows that $\sigma\Pi_1 z = \sigma x \in \langle \sigma, \tau'\rangle$.

$\geq$. It suffices to show that for all $\tau : T \to A^\perp/x$ and $y \in \langle\sigma, \tau\rangle$ there exists a $+$-maximal $x \in \mathcal{C}(S)$ such that $\kappa(\sigma x) \leq \kappa(y)$. But for all such y there is $z \in \mathcal{C}(S) \odot \mathcal{C}(T)$ maximal such that $y = \sigma\Pi_1 z$. Set $x = \Pi_1 z$, since z is maximal x must be $+$-maximal, and $\kappa(\sigma x) = \kappa(y)$. □

Proposition E.2 *For strategies $\sigma : S \to A^\perp \parallel B$ and $\tau : T \to B^\perp \parallel C$, we have $v^\downarrow(\tau \odot \sigma) \geq v^\downarrow(\tau) + v^\downarrow(\sigma)$. Likewise for $\sigma : S \to A$ and $\tau : T \to B$, we have $v^\downarrow(\sigma \parallel \tau) = v^\downarrow(\sigma) + v^\downarrow(\tau)$.*

Proof. Straightforward using Proposition E.1. □

Proposition E.3 *For any race-free, well-founded games A and B, we have $v(A \parallel B) = v(A) + v(B)$.*

Proof. By the value theorem, it does not matter whether we work on the optimistic or pessimistic cases. By simplicity, let us pick the pessimistic one. Firstly, we prove that $v(A \parallel B) \geq v(A) + v(B)$. Indeed, let $\sigma : S \to A$ and $\tau : T \to B$ be strategies. Then, as needed we have $v^\downarrow(\sigma \parallel \tau) \geq v^\downarrow(\sigma) + v^\downarrow(\tau)$ by Proposition E.2.

Moreover, this inequality also holds for $A^\perp$ and $B^\perp$, therefore $v(A^\perp \parallel B^\perp) \geq v(A^\perp) + v(B^\perp)$, from which it follows that $v(A \parallel B) \leq v(A) + v(B)$ by the value theorem and the definition of the dual of games with payoff. □

Theorem E.4 *If $\sigma : S \to A^\perp \parallel B$ and $\tau : T \to B^\perp \parallel C$ are optimal strategies, so is $\tau \odot \sigma$. Moreover copycat is optimal, therefore there is a bicategory of concurrent games with payoff and optimal strategies.*

Proof. Suppose σ and τ optimal. We reason as follows:

$$\begin{aligned} v^\downarrow(\tau \odot \sigma) &\geq v^\downarrow(\sigma) + v^\downarrow(\tau) \\ &= v(A^\perp \parallel B) + v(B^\perp \parallel C) \\ &= v(A^\perp \parallel C) \end{aligned}$$

This implies that $v^\downarrow(\tau \odot \sigma) = v(A^\perp \parallel C)$, since a strict inequality would contradict the definition of $v(A^\perp \parallel C)$.

Copycat is optimal: take a $+$-maximal $x \in \mathcal{C}(A^\perp \parallel A)$. Necessarily, x has the form $y \cup \overline{y}$, where $y \in \mathcal{C}(A)$. Moreover, $\kappa_{A^\perp \parallel A}(x) = \kappa_A(y) - \kappa_A(y) = 0$, therefore by Proposition E.1, we have $v^\downarrow(\gamma_A) = 0$. However we also have $v(A^\perp \parallel A) = v(A) - v(A) = 0$, therefore copycat is optimal. □

Nominal Lambda Calculus: An Internal Language for FM-Cartesian Closed Categories

Roy L. Crole and Frank Nebel

Dept of Computer Science, University of Leicester, University Road, Leicester, LE1 7RH, U.K.

Abstract

Reasoning about atoms (names) is difficult. The last decade has seen the development of numerous novel techniques. For equational reasoning, Clouston and Pitts introduced Nominal Equational Logic (NEL), which provides judgements of equality and freshness of atoms. Just as Equational Logic (EL) can be enriched with function types to yield the lambda-calculus (LC), we introduce NLC by enriching NEL with (atom-dependent) function types and abstraction types. We establish meta-theoretic properties of NLC; define ИFM-cartesian closed categories, hence a categorical semantics for NLC; and prove soundness & completeness by way of NLC-classifying categories. A corollary of these results is that NLC is an internal language for ИFM-cccs. A key feature of NLC is that it provides a novel way of encoding freshness assertions, and a new vehicle for studying the interaction of freshness together with higher order types.

Keywords: category theory, dependent types, FM-sets, internal language, nominal logic, semantics, type theory

1 Introduction

Categorical gluing is a fascinating topic. Crole has had a long-standing interest, especially in the connections with logical relations. In [9] Crole developed categorical logical relations and showed that the relational glued category simplifies the Freyd scone [13], and applied the construction to prove that the $(\beta\eta)\lambda$-calculus is conservative over algebraic theories (and more besides). The starting point for our current work was to ask if there is a notion of nominal categorical logical relation which could be used to prove conservative extension results—for the type theory we considered Nominal Equational Logic (NEL).

(NEL) was introduced by Clouston and Pitts in [8] (closely related to Nominal Algebra introduced by Gabbay and Mathijssen [16]). Space forces us to assume familiarity with NEL, but here is a quick overview: NEL extends equational logic EL [10,24] (where types denote ZF-sets). NEL variables are thought of as elements of FM-sets (roughly speaking, sets whose elements have a finite support of atoms/(names) in the sense of Gabbay and Pitts [14] and for which one can make

[1] Email: r.crole@mcs.le.ac.uk

assertions about the freshness of atoms). The motivation for NEL is to provide a system for formal equational reasoning *combined* with reasoning about the freshness of atoms—the latter an important topic of study in Programming Semantics. To this end one seeks a theory with a sound and complete semantics. The theory must necessarily capture permutation actions, finite support, and freshness. As such, one might expect to be able to make judgements $a \# M$, asserting atom a is fresh for M, as well as $M = M'$. Further, we need to be able to assert hypotheses $a \# x$ about variables x that may occur (freely) in M. Indeed in NEL one sees $\overline{a}_1 \# x_1 : s_1, ..., \overline{a}_n \# x_n : s_n \vdash^{\#} \overline{a} \# M : s$ capturing the intuition that if sets of atoms $\overline{a}_i$ are fresh for (the interpretation of) the x_i, then $\overline{a}$ is fresh for M. One might also work instead with judgements $\overline{a}_1 \# x_1 : s_1, ..., \overline{a}_n \# x_n : s_n \vdash^{E} M : s$ and then codify $\overline{a} \# M$ by way of an equation, since freshness can be defined equationally [6] (under suitable conditions). This is the approach we take.

Clouston has shown in [5,7] that the category *FMSet* provides a sound and complete semantics for NEL. Further he defines the notion of an FM-category, axiomatising the underlying structure of *FMSet*, and shows that such categories yield a sound semantics. He shows that a NEL theory has a classifying FM-category in which there is a generic model [10,31]—hence his semantics is also complete. Indeed, Clouston shows that there is a correspondence between NEL theories and FM-categories establishing that NEL is an internal language for FM-categories. Lambek [21] showed that theories in the λ-calculus correspond to cartesian closed categories (a proof using functional completeness, with Scott, appears in [22]; see also [10]).

A natural question to ask is whether there is a notion of nominal λ-calculus (NLC) that corresponds to some form of "cartesian closed FM-category". Moreover, if there is, we can test the robustness of both NEL, and the methodology of categorical logical relations, by attempting to show that NLC is conservative over NEL using gluing. To do this we need to develop NLC and a suitable categorical correspondence, which we do in this paper.

Before we begin the task at hand, we justify our overall approach. At the conceptual level, this paper concerns itself with the fascinating notion of correspondences between category theory and type theory. This arises from Lawvere's seminal work [23]. There are two approaches that one could take in formulating such correspondences. (i) is to demonstrate that models of a theory Th in a category $\mathcal{C}$ (and maps between models), $Mod(Th, \mathcal{C})$, correspond to structure preserving functors (and natural transformations between functors), $SPF(Cl(Th), \mathcal{C})$. (ii) is to show the existence of a monad T_{Th} for which $Mod(Th, \mathcal{C})$ corresponds to the (Eilenberg-Moore) algebras of T_{Th}. Both approaches have their merits. For some deep insights into the heart of the matter in the case of theories in equational logic consult Hyland and Power's overview [19]. An elegant approach via monads, providing a very general framework, is established in the work of Berger, Mèlliés and Weber [2] and Mèlliés [26]. However, for computer science and (foundations of) program semantics, where one may well be seeking a rigorously specified syntactic type theory capable of being formalised, approach (i) seems to be the path to follow (please see Section 7 for additional commentary). In particular, we want to establish that any such theory is indeed the internal language of a suitable category with structure, with the usual adjunction $Cl \dashv Th$.

Remark 1.1 The category central to this work is *FMSet* [5,15]. The category of nominal sets *FMNom* is relevant too: for a very clear introduction see [30]. While the properties of *FMSet* are less well known than *FMNom*, both are toposes $\mathcal{T}$. As such each is equipped with a Higher Order Logic internal language $Th_{\mathcal{T}}$. Thus one might ask whether one could automatically capture the notion of FM-ccc by internalisation of cartesian closure (and freshness) in $\mathcal{T}$; and indeed "extract" NLC from the HOL $Th_{\mathcal{T}}$, perhaps by extending $Th_{\mathcal{T}}$ with additional axioms. It is not clear to us that this can be done, or, if it can, whether it it can circumvent the detail in this paper bearing in mind that our aim from the "computer science" perspective is to produce a fully formalised type theory. See Section 7 for more discussion.

We build directly on [7], taking approach (i). We have tried to keep this paper as self-contained as possible, but cannot include all of the definitions and lemmas for lack of space. Some are included in an appendix. In Section 2 we specify the types and terms of NLC without abstraction. We define permutation actions, capture avoiding substitution, and α-equivalence. We prove results about the terms which we will use when proving soundness and completeness. In Section 3 we specify the NLC type system and define NLC equational theories, without abstraction. We again prove key results for soundness and completeness. In Section 4 we introduce FM-cartesian closed categories, showing they soundly model NLC without abstraction. In Section 5 we add abstraction and concretion to NLC and show that our semantics is sound and complete for ИFM-cccs, which are FM-cccs with additional structure that models abstraction and concretion. In Section 6 we show that ИFM-cccs are syntax free presentations of NLC theories. In Section 7 we discuss applications and further work. Please see the Results Road Map on page 102. Here are the main contributions:

- Higher order functions that naturally extend NEL are partial in the sense that their arguments must satisfy freshness conditions. We believe that this is the first paper to posit a move to a "types dependent on atoms" type theory in order to capture, in a novel type system, this partiality of higher order functions (see page 88 for details). NLC allows us to examine the combination of the freshness relation and higher types in a new light.
- Dependent types enable us to to specify name abstraction and concretion. The operation of concretion is inherently partial, and indeed cannot be captured as a NEL theory—see Clouston [4]. However NLC dependent types do provide a mechanizm to capture this partiality.
- In [7] the type system for terms is separate from the system for freshness assertions, (a two part type system). Moreover typing judgements predicated on freshness assertions are not first class citizens (but simply reflexive equations). We introduce rules (see page 102) for a single first class type system. This is not only necessitated by the dependent types, but significantly simplifies and unifies the judgements forms in [7].
- A clean formulation of a categorical semantics of NLC. The semantics is considerably complicated by both type dependency on atoms, and the encapsulation

of freshness judgements by equational axioms. Our single first class type system simplifies our soundness proof from what it would otherwise have been.

- A simplification of Clouston's meta-theory [7]. We show that all the key properties of (syntactic) permutation actions we require can be defined cleanly on raw terms, prior to type-checking. This material is mainly in Section 2 and the appendix.
- A detailed proof of an "approach (i)" category theory type theory correspondence, yielding NLC theories as the internal language of FM-cccs, and hence completeness. We pay very careful attention to details that are significant for implementations (see for example the proof on page 107 of Lemma 3.3).

We will use the following notation: Let $\mathbb{A}$ be the set of atoms (names). We write $\overline{a}$ or similar for typical finite subsets $\{a_1, \ldots, a_k\}$ of $\mathbb{A}$. We write π or similar for any permutation on $\mathbb{A}$ with finite domain. *Perm* denotes the set of such permutations (equivalently those generated by transpositions $(a\, b)$). The composition of π and π', with π' acting first, is denoted by $\pi \circ \pi'$ or $\pi\pi'$. If $X = (X, \cdot)$ is an FM-set, and $x \in X$, we write $supp(x)$ for the support of x, and $\overline{a} \# x$ to denote that each atom in $\overline{a}$ is not in $supp(x)$.

2 The Meta-Theory of NLC Terms without Abstraction

Remark 2.1 Until Section 5 we work with a subset of NLC. This will allow us to fully motivate the use of a form of dependent typing in order to formulate our extension of NEL with higher order functions. Abstraction and concretion is omitted until later in the paper.

In NEL one works with a nominal set of types [2] . In NLC we work with a nominal set of ground types, and generate the function types. NLC extends NEL terms with function abstractions and applications. An abstraction takes the form $\lambda^{\overline{a}} x : s.\, M$ and we explain the intended semantic interpretation. In NEL we may have $\overline{a} \# x : s \vdash^E M : s'$. If we want to capture the "mapping" $x \mapsto M$ as an abstraction, we could consider $\lambda\, x : s.M$. However, if we apply $\lambda\, x : s.M$ to a term $N : s$ we also need to ensure that $\overline{a} \# N$. We might codify the set $\overline{a}$ in the abstraction $\lambda^{\overline{a}} x : s.\, M$. So far so good. But what about types? In NEL, the FM-set semantics of $\overline{a} \# x : s$ is specified by requiring that $[\![x]\!] \in [\![s]\!]^{\# \overline{a}} \overset{\text{def}}{=} \{e \in [\![s]\!] \mid \overline{a} \# e\}$. So one might wonder if $s^{\overline{a}}$ could be be a suitable type for the source of our abstraction, with a compositional semantics $[\![s^{\overline{a}}]\!] \overset{\text{def}}{=} [\![s]\!]^{\# \overline{a}}$. We can then consider the type $s^{\overline{a}} \Rightarrow s'$ for our abstraction, hoping that if our semantics is defined in a compositional way, it will have all of the relevant equivariance and categorical properties to yield a sound and complete semantics. This abstraction typing is deceptively simple: the type and equation system that results is intuitive, but quite complex to manipulate since function types now depend on atoms.

NLC-*Signatures, Types, and Raw Terms.* We start with an analogue of the notion of a signature for λ-calculus. A NLC-signature *Sg* is specified by

[2] In [7] "types" are called sorts. We use the word type since it better matches general usage in computer science, and categorical type theory

(i) Gnd_{Sg}, a nominal **set of ground types**. The **set of types** $Type_{Sg}$ is then generated by the BNF grammar $s ::= \gamma \mid s^{\overline{a}} \Rightarrow s$ where γ is any ground type. Since each type s is a finite tree and Gnd_{Sg} is a nominal set, each s is finitely supported with the permutation action

$$\pi \cdot \gamma \overset{\text{def}}{=} \pi \cdot_{Gnd_{Sg}} \gamma \qquad \pi \cdot (s^{\overline{a}} \Rightarrow s') \overset{\text{def}}{=} (\pi \cdot s)^{\pi \cdot \overline{a}} \Rightarrow (\pi \cdot s')$$

and hence $Type_{Sg}$ is a nominal set of types.

(ii) A nominal **set of (higher order function) constant symbols** Fun_{Sg}.

(iii) An equivariant typing function $Fun_{Sg} \to Type_{Sg}$, which assigns to each constant symbol c a type; we refer to a **typing** $c : s$.

Fixing a set $Var \overset{\text{def}}{=} \{\mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3, \ldots\}$ of (ordered) **variables**, the raw NLC-terms are specified by $M := \pi x \mid c \mid \lambda^{\overline{a}} x : s.\, M \mid M\ M$ where πx is a **suspension** [8,7] of any variable $x \in Var$. We refer to the set of raw terms for signature Sg by $Term_{Sg}$. Variables may be **free** or **bound** (where all occurrences of x in any "subterm" $\lambda^{\overline{a}} x : s.\, M$ are bound).

Permutation Actions for Raw Terms. Recall [8,16] the two standard permutation actions on *Perm*, namely conjugation (which is finitely supported) and left multiplication (which is not). Clouston & Pitts and Gabbay & Mathijssen define two permutation actions, called meta-level $\pi \cdot M$ and object-level $\pi * M$ [8,7], which are syntactic analogues of the actions on *Perm*. In categorical type theory one always works with terms in context. As such, a term M with a free variable x is always regarded as a "function" $x \mapsto M$. The permutation action on functions found in nominal and FM-sets is a (form of) conjugation action and the syntactic analogue is $\pi \cdot M$. However it is useful to work also with a simple action in which π acts on M simply by acting recursively over the structure of a term: eg $\pi * (\tau x)(\tau' y) = (\pi * \tau x)(\pi * \tau' y) = (\pi\tau x)(\pi\tau' y)$.

We define such actions for NLC. To do so, consider the recursive definition of mappings $(\pi, M) \mapsto \pi * M$ and $(\pi, M) \mapsto \pi \cdot M$ in Table 1. Note that in order to define the object-level permutation we first define a basic form of substitution $M[\pi^{-1}x/x]$, on raw terms M. We call this a **suspension-substitution**. Informally, free occurrences of x in M are replaced by $\pi^{-1}x$. Formally, the recursive definition is the expected one, where on suspensions we define $(\pi' y)[\pi^{-1}x/x] \overset{\text{def}}{=} \pi' y$ if $x \neq y$ and $(\pi' x)[\pi^{-1}x/x] \overset{\text{def}}{=} (\pi'\pi^{-1})x$.

To show in Proposition 2.2 that the mappings in Table 1 are permutation actions, we need Lemma C.1 (see Appendix).

Proposition 2.2 (Permutation Action Definitions)

- *The mapping $(\pi, M) \mapsto \pi \cdot M$ is a permutation action; we call it the* ***meta-level*** *permutation action. It is finitely supported, so the set $Term_{Sg}$ of raw* NLC*-terms is a nominal set. The finite support of a raw term is specified recursively where $supp(\pi x) \overset{\text{def}}{=} supp(\pi)$, $supp(\lambda^{\overline{a}} x : s.\, M) \overset{\text{def}}{=} \overline{a} \cup supp(s) \cup supp(M)$ and $supp(M\ N) \overset{\text{def}}{=} supp(M) \cup supp(M)$; and constants are finitely supported by definition.*

• $\pi \cdot \pi' x \overset{\text{def}}{=} (\pi\pi'\pi^{-1})x$	• $\pi * \pi' x \overset{\text{def}}{=} (\pi\pi')x$
• $\pi \cdot c \overset{\text{def}}{=} \pi \cdot_{Fun_{Sg}} c$	• $\pi * c \overset{\text{def}}{=} \pi \cdot_{Fun_{Sg}} c$
• $\pi \cdot (\lambda^{\overline{a}} x : s.M) \overset{\text{def}}{=} \lambda^{\pi \cdot \overline{a}} x : \pi \cdot s.(\pi \cdot M)$	• $\pi * (\lambda^{\overline{a}} x : s.M) \overset{\text{def}}{=}$ $\lambda^{\pi \cdot \overline{a}} x : \pi \cdot s.(\pi * (M[\pi^{-1} x/x]))$
• $\pi \cdot (M\ N) \overset{\text{def}}{=} (\pi \cdot M)\ (\pi \cdot N)$	• $\pi * (MN) \overset{\text{def}}{=} (\pi * M)(\pi * N)$
Meta-Level	Object-Level

Table 1
Permutation Actions for NLC

- *The mapping $(\pi, M) \mapsto \pi * M$ is a permutation action; we call it the* ***object-level*** *permutation action.*

Capture Avoiding Substitution and α-Equivalence. We require simultaneous capture-avoiding substitution of raw terms. This will be crucial for defining composition of morphisms in a classifying category–see Proposition 5.1. Since the high level ideas of this paper can be read without recourse to such detail, we simply state our notation here, and refer the reader to Section C for the details (our approach simplifies Clouston's [7]). Substituting $N_1, \ldots, N_n$ for free occurrences of the distinct variables $x_1, \ldots, x_n$ in the raw term M yields another raw term, which we denote by $M\{N_1, \ldots, N_n/x_1, \ldots, x_n\}$ or by $M\{\boldsymbol{N_i}/\boldsymbol{x_i}\}$.

So far we have used structural equality on terms $M = N$. Since we wish to work with capture avoiding substitution (to construct our classifying category) which makes use of variable renaming, we have to replace $=$ with α-equivalence $\sim_\alpha$. It can easily be shown that $\sim_\alpha$ is equivariant for the permutation actions, that is $M \sim_\alpha N$ implies $\pi \cdot M \sim_\alpha \pi \cdot N$ and respectively for the object level permutation action. From this a well-defined permutation action on α-equivalence classes of terms is induced: $\pi \cdot [M]_\alpha \overset{\text{def}}{=} [\pi \cdot M]_\alpha$ and $\pi * [M]_\alpha \overset{\text{def}}{=} [\pi * M]_\alpha$. Capture avoiding substitution lifts to the set of α-equivalence classes of terms, $Term_{Sg}/_{\sim_\alpha}$.

Remark 2.3 We call $[M]_\alpha$ an **expression**. Having taken great care in defining expressions $[M]_\alpha$ (a key component of our work; details in the Appendix page 105), we adopt the usual convention of writing just M. However, all our proofs deal correctly with the intricacies that arise from variable re-naming to avoid capture (see for example [28] (page 169) and [25]).

The next propositions are crucial for our main theorems, the first ($*$ associates with $\{/\}$) by induction on M, the second ($*$ distributes over $\{/\}$) by direct calculation being a corollary of Proposition 2.4 and Lemma C.3. For expressions $[M]_\alpha$, distinct variables $x_1, \ldots, x_n$, and expressions $[N_1]_\alpha, \ldots, [N_n]_\alpha$ we have

Proposition 2.4 $(\pi * [M]_\alpha)\{\boldsymbol{[N_i']_\alpha}/\boldsymbol{x_i}\} = \pi * ([M]_\alpha\{\boldsymbol{[N_i']_\alpha}/\boldsymbol{x_i}\})$

Proposition 2.5 $\pi \cdot (M\{\boldsymbol{N_i}/\boldsymbol{x_i}\}) = (\pi \cdot M)\{\boldsymbol{(\pi \cdot N_i)}/\boldsymbol{x_i}\}$ *(Written using the convention for α-equivalence classes, generally adopted from now on.)*

3 NLC Typed Expressions and Equational Theories

We define NLC by specifying a type and equation system. The intuitions of NLC and NEL are the same, but technicalities are quite different. In NEL, terms are typed using environments $\Gamma \overset{\text{def}}{=} x_1 : s_1, \dots, x_n : s_n$, just like ordinary equational logic. The judgements either take the form $\Gamma \vdash M : s \quad (\diamond)$, or $\nabla \vdash^E M \approx M' : s$ where $\nabla = \overline{a}_1 \# x_1 : s_1, ..., \overline{a}_n \# x_n : s_n$ records assumptions about freshness and types. NEL judgements $\nabla \vdash^E M : s$ are simply sugar for reflexive equations. The type system $(\diamond)$ is entirely separate from the freshness system (in two parts)! We found this slightly confusing. Indeed, with NLC we cannot separate the type system in this way, since the types of abstractions depend directly on freshness assertions. Thus the environments used in the type system must encode freshness assertions (and cannot be of the form Γ)! Our typing judgements $\nabla \vdash^E M : s$ are first class citizens (in a single system). They are not abbreviations for reflexive equations. This is not merely dabbling with unnecessary cosmetic idolatry: it simplifies the presentation of our categorical semantics and is a key contribution.

Recall the formal notion of a freshness environment [7] (included below). We can then define expressions, and equations, in context and finally present the NLC type and equation systems.

A **freshness environment**, or just **environment**, is a finite partial function $\nabla : \mathit{Var} \to \mathcal{P}_{fin}(\mathbb{A}) \otimes \mathit{Type}_{Sg}$ with finite domain. By definition it maps each $x \in \mathit{dom}(\nabla)$ to a pair $(\overline{a}, s)$ where $\overline{a}$ is a finite set of atoms $s \in \mathit{Type}_{Sg}$ and $\overline{a} \# s$. The set of environments Env_{Sg} is a nominal set under the permutation action $(\pi \cdot \nabla)(x) = (\pi \cdot \overline{a}, \pi \cdot s)$. We often write an environment ∇ as $\overline{a}_1 \# x_1 : s_1, ..., \overline{a}_n \# x_n : s_n$. For ∇, ∇', we write $\nabla \leq \nabla'$ if $\mathit{dom}(\nabla) \subseteq \mathit{dom}(\nabla')$ and for all $x \in \mathit{dom}(\nabla)$ we have $\mathit{pr}_1(\nabla(x)) \subseteq \mathit{pr}_1(\nabla'(x))$ and $\mathit{pr}_2(\nabla(x)) = \mathit{pr}_2(\nabla'(x))$.

- We define an **expression-in-context** as a judgement of the form $\nabla \vdash^E M : s$ where ∇ is a freshness environment, M is an α-equivalence class of NLC-terms (an expression) and s is a type.
- An **equation-in-context** is a judgement of the form $\nabla \vdash^E M \approx M' : s$ where $\nabla \vdash^E M : s$ and $\nabla \vdash^E M' : s$.

A NLC-theory Th is a pair $(\mathit{Sg}, \mathit{Ax})$, where Sg is a NLC-signature and Ax is a collection of equations-in-context. We shall use Th to inductively define a subset of expressions-in-context and equations-in-context. Any expression-in-context that has a derivation is a **typed expression**; and any such equation-in-context is a **theorem**. The set of typed expressions and theorems of a NLC-theory Th is the least set of judgements containing the axioms of Th and closed under the rules in Table A.1 in Appendix A. We indicate that any judgement J has a derivation in theory Th by writing $\mathit{Th} \rhd J$.

Remark 3.1 Justified by [6] we use the following abbreviation: for $\nabla \vdash M : s$ and $\overline{a} \subseteq \mathbb{A}$ $(\overline{a} \# s)$, we write $\nabla^{\# \overline{z}} \overset{\text{def}}{=} \overline{a}_1 \cup \overline{z} \# x_1 : s_1, ..., \overline{a}_n \cup \overline{z} \# x_n : s_n$ and

$$\nabla \vdash^E \overline{a} \# M : s \overset{\text{def}}{=} \nabla^{\# \overline{z}} \vdash^E M \approx (\overline{a}\, \overline{z}) * M : s.$$

In the transposition, $\overline{a} \in \mathbb{A}^n$ is sugar for a tuple of the atoms in the set $\overline{a}$ and

$\overline{z} \in \mathbb{A}^n$ is any/some fresh tuple of the same size such that $\overline{z} \# (\nabla, \overline{a}, M)$. If $Th \rhd \nabla \vdash^E \overline{a} \# M : s$ then we may legitimately call the judgement a theorem, but we will usually call it a **freshness assertion**.

The role that the judgements $\nabla \vdash^E \overline{a} \# M : s$ play leads to a crucial difference between NEL and NLC. Consider the rule AP. Since F has type $s^{\overline{a}} \Rightarrow s'$ then $\overline{a}$ must be fresh for the argument A, formally encoded as $\nabla \vdash^E \overline{a} \# A : s$. Thus the type system rules have equations-in-context as hypotheses, and the equation rules have expressions-in-context as hypotheses. Thus theorems and typed expressions are mutually inductively defined. Obviously this complicates our proofs, at least in comparison to NEL, and leads to some subtleties which we explain in due course.

We have two more lemmas that are crucial for proving some important facts about NLC. Lemma 3.2 is used in induction steps in which a binding variable in an abstraction also occurs in the environment (of the abstraction): For an example induction see the Appendix proof (page 107) of Lemma 3.3, and [28] (page 169) for a detailed explanation of the problem. Lemma 3.3 is used in proving Proposition 3.4; the proposition underpins our semantics and classifying category construction.

Lemma 3.2 (Variable Equivariance of Judgements) *All typed expressions, and all theorems (hence freshness assertions too), are equivariant under variable swapping. More precisely, for any two distinct variables x, y, and where $(x\,y) \bullet -$ denotes variable swapping (see page 105)*

$$Th \rhd \nabla \vdash^E M : s \Longrightarrow Th \rhd (x\,y) \bullet \nabla \vdash^E (x\,y) \bullet M : s$$
$$Th \rhd \nabla \vdash^E M \approx M' : s \Longrightarrow Th \rhd (x\,y) \bullet \nabla \vdash^E (x\,y) \bullet M \approx (x\,y) \bullet M' : s$$

Lemma 3.3 *$Th \rhd \nabla, \overline{a} \# x : s \vdash^E M : s'$ if and only if $Th \rhd \nabla, \pi \cdot \overline{a} \# x : \pi \cdot s \vdash^E M\{\pi^{-1}x/x\} : s'$ and similarly for equations.*

In order to define our categorical semantics, we will require Proposition 3.4 and Proposition 3.5. Some example proof details are in the Appendix.

Proposition 3.4 ($*$ preserves Typed Expressions and "Equalities") *Given a theory Th,*

$$Th \rhd \nabla \vdash^E M : s \textit{ implies } Th \rhd \nabla \vdash^E \pi * M : \pi \cdot s$$
$$Th \rhd \nabla \vdash^E M \approx M' : s \textit{ implies } Th \rhd \nabla \vdash^E \pi * M \approx \pi * M' : \pi \cdot s$$

Proposition 3.5 (Atom Equivariance of Judgements) *Given a theory Th,*

$$Th \rhd \nabla \vdash^E M : s \textit{ implies } Th \rhd \pi \cdot \nabla \vdash^E \pi \cdot M : \pi \cdot s$$
$$Th \rhd \nabla \vdash^E M \approx M' : s \textit{ implies } Th \rhd \pi \cdot \nabla \vdash^E \pi \cdot M \approx \pi \cdot M' : \pi \cdot s$$

4 A Sound Categorical Semantics

FM-Cartesian Closed Categories. Underlying intuition for FM-cccs starts by considering internal categories $\mathcal{I}$ in *FMNom*. Such structures, while necessary for modelling NLC, are not sufficiently rich: to give meaning to NLC terms we must encode permutation actions as morphisms—an additional requirement on $\mathcal{I}$. We follow the "type (i) approach": axiomatising $\mathcal{I}$ externally and equipping with permutation morphisms, yields *a category with finitely supported internal permutation actions*. We then obtain good notions of products and exponentials by stipulating coherence conditions between these structures and the internal permutation action; these are *cartesian closed perm-categories*. The (additional, external) axiomatisation of freshness properties yields *FM-cccs*. Further details of FM-categories are in [7].

A category $\mathcal{C}$ has an **internal permutation action** if for each $\pi \in \mathit{Perm}$ and $C \in \mathit{ob}\,\mathcal{C}$ there is a $\mathcal{C}$-arrow $\pi_C : C \to \pi \cdot C$ such that ι_C is the identity id_C and $(\pi' \circ \pi)_C = \pi'_{\pi \cdot C} \circ \pi_C$, where $\pi \cdot C$ is defined to be the codomain of π_C. An internal permutation action is **finitely supported** if every arrow $f : C \to D$ in $\mathcal{C}$ is finitely supported with respect to the permutation action $\pi \cdot f \overset{\text{def}}{=} \pi_D \circ f \circ (\pi^{-1})_{\pi \cdot C}$. We call a category with a finitely supported permutation action a **perm-category**. A perm-category has **equivariant products** if it has finite products, and the internal permutation action preserves the projections (hence also preserves the product objects). A perm-category with equivariant finite products has **equivariant exponentials** if it is cartesian closed and the internal permutation action preserves the evaluation morphism $\pi \cdot \mathit{ev}_{A,B} = \mathit{ev}_{\pi \cdot A, \pi \cdot B}$ (and hence exponential objects are preserved). A perm-category with equivariant finite products has **fresh inclusions** if for every finite set of atoms $\overline{a} \subseteq \mathbb{A}$ and $\mathcal{C}$-object C such that $\overline{a} \# C$ we have a $\mathcal{C}$-arrow $i^{\overline{a}}_C : C^{\#\overline{a}} \to C$ for which the following properties hold:

(i) (Equivariance): $\pi \cdot i^{\overline{a}}_C = i^{\pi \cdot \overline{a}}_{\pi \cdot C}$;

(ii) (Sets of Atoms): $i^{\emptyset}_C = \mathit{id}_C$ and $i^{\overline{a}}_C \circ i^{\overline{a'}}_{C^{\#\overline{a}}} = i^{\overline{a} \cup \overline{a'}}_C$;

(iii) (Products): $i^{\overline{a}}_{C_1 \times C_2} = i^{\overline{a}}_{C_1} \times i^{\overline{a}}_{C_2}$;

(iv) (Internal permutation action): if $\mathit{supp}(\pi) \# C$ then $\pi_{C^{\#\mathit{supp}(\pi)}}$ is equal to the identity $\mathit{id}_{C^{\#\mathit{supp}(\pi)}}$;

(v) (Epi When Fresh): If we have parallel $\mathcal{C}$-arrows $f, g : C \to D$ such that $f \circ i^{\overline{a}}_C = g \circ i^{\overline{a}}_C$ and $\overline{a} \# (f, g)$, then $f = g$;

(vi) (Freshness): Let $f : C \to D$ be such that $\overline{a} \# D$. Define $\dagger(f, \overline{a}) \overset{\text{def}}{=} (\exists\, \overline{b})(\overline{b} \# (\overline{a}, f) \wedge (\overline{a}\, \overline{b})_D \circ f \circ i^{\overline{b}}_C = f \circ i^{\overline{b}}_C)$. If $\dagger(f, \overline{a})$ holds then there is a unique $f^* : C \to D^{\#\overline{a}}$, the **image restriction** of f, such that $i^{\overline{a}}_D \circ f^* = f$.

A perm-category with equivariant finite products and fresh inclusions is an **FM-category** and if it also has equivariant exponentials we call it an **FM-ccc**. The category *FMSet* of FM-sets is an FM-ccc, with the (equivariant) exponential of FM-sets X and Y being the FM-set $X \Rightarrow_{fs} Y$ of finitely supported functions from X to Y, and with $i^{\overline{a}}_X : X^{\#\overline{a}} \overset{\text{def}}{=} \{x \in X \mid \overline{a} \# x\} \hookrightarrow X$ as fresh inclusions. FM-cpos are another example.

Remark 4.1 Each freshness property has a simple intuition. We give one example for (Freshness). Let $f : X \to Y$ be finitely supported in *FMSet*, $x \in X$ and $a \# Y$. By choosing $b \# a, f$ *and* $b \# x$ we have $(f \circ i^b_C)(x) = f(x)$ and the condition $\dagger(f, a)$ amounts to $(b \# a, f) \wedge (a\, b) \cdot f(x) = f(x)$. But since $b \# x$ we can also deduce $b \# f(x)$, so we have $(b \# a, f(x)) \wedge (a\, b) \cdot f(x) = f(x)$. Hence $f(x) \in Y^{\#a}$ and so f image restricts (with $f^* : x \mapsto f(x)$).

$$\llbracket \nabla, \overline{a_i} \# x_i : s_i \vdash \pi x_i : \pi \cdot s_i \rrbracket \blacktriangleright$$
$$\pi_{\llbracket s_i \rrbracket} \circ i^{\overline{a_i}}_{\llbracket s_i \rrbracket} \circ pr_i : \llbracket \nabla, \overline{a_i} \# x_i : s_i \rrbracket \longrightarrow \llbracket s_i \rrbracket^{\# \overline{a_i}} \longrightarrow \llbracket s_i \rrbracket \longrightarrow \pi \cdot \llbracket s_i \rrbracket$$

$$\llbracket \nabla \vdash c : s \rrbracket \blacktriangleright \llbracket c \rrbracket \circ ! : \llbracket \nabla \rrbracket \to 1 \to \llbracket s \rrbracket$$

$$\frac{\llbracket \nabla, \overline{a} \# x : s \vdash M : s' \rrbracket \blacktriangleright m : \llbracket \nabla \rrbracket \times \llbracket s \rrbracket^{\overline{a}} \to \llbracket s' \rrbracket}{\llbracket \nabla \vdash \lambda^{\overline{a}} x : s.M : s^{\overline{a}} \Rightarrow s' \rrbracket \blacktriangleright \lambda(m) : \llbracket \nabla \rrbracket \to (\llbracket s \rrbracket^{\# \overline{a}} \Rightarrow \llbracket s' \rrbracket)}$$

$$\frac{\llbracket \nabla \vdash F : s^{\overline{a}} \Rightarrow s' \rrbracket \blacktriangleright f : \llbracket \nabla \rrbracket \to (\llbracket s \rrbracket^{\# \overline{a}} \Rightarrow \llbracket s' \rrbracket) \quad \llbracket \nabla \vdash \overline{a} \# A : s \rrbracket \blacktriangleright \theta : \llbracket \nabla \rrbracket \to \llbracket s \rrbracket^{\# \overline{a}}}{\llbracket \nabla \vdash F\ A : s' \rrbracket \blacktriangleright ev \circ \langle f, \theta \rangle : \llbracket \nabla \rrbracket \to (\llbracket s \rrbracket^{\# \overline{a}} \Rightarrow \llbracket s' \rrbracket) \times \llbracket s \rrbracket^{\# \overline{a}} \to \llbracket s' \rrbracket}$$

$$\frac{\llbracket \nabla \vdash^E M : s \rrbracket \blacktriangleright m}{\llbracket \nabla \vdash^E \overline{a} \# M : s \rrbracket \blacktriangleright m^*} \quad \dagger(m, \overline{a})$$

Table 2
Semantics in an FM-ccc

A Sound Categorical Semantics. We wish to define a categorical semantics which will interpret typed expressions $Th \rhd \nabla \vdash^E M : s$ as morphisms $\llbracket \nabla \vdash^E M : s \rrbracket : \llbracket \nabla \rrbracket \longrightarrow \llbracket s \rrbracket$ in an FM-ccc $\mathcal{C}$. However we have seen that NLC is dependently typed: in particular the type system and equation system are mutually inductively defined. This means that we cannot give a simple recursive definition of a function $\llbracket - \rrbracket$ over (well-typed) expressions [31,33]. However, we can give such a definition of a partial semantic function, which is defined only when certain equations are themselves satisfied by $\llbracket - \rrbracket$.

We also deal with a further complication. See rule AP which has hypothesis $\nabla \vdash^E \overline{a} \# A : s$. We wish to define, following Remark 4.1, the semantics of $\nabla \vdash^E \overline{a} \# A : s$ as $\llbracket \nabla \vdash^E \overline{a} \# A : s \rrbracket \overset{\text{def}}{=} \llbracket \nabla \vdash^E A : s \rrbracket^*$—but this morphism is defined only if the condition $\dagger(\llbracket \nabla \vdash^E A : s \rrbracket, \overline{a})$ holds! Thus we also need to factor this requirement into our semantics and soundness theorem.

We can now define the semantics. Let $\mathcal{C}$ be a FM-ccc and Sg a NLC-signature. Then a Sg-**structure** $\mathcal{M}$ in $\mathcal{C}$ is specified by giving:

- An equivariant map $\llbracket - \rrbracket : Gnd_{Sg} \longrightarrow ob\,\mathcal{C}$. We extend to the map $\llbracket - \rrbracket : Type_{Sg} \longrightarrow ob\,\mathcal{C}$ via structural recursion ($\llbracket s^{\overline{a}} \Rightarrow s' \rrbracket \overset{\text{def}}{=} \llbracket s \rrbracket^{\# \overline{a}} \Rightarrow \llbracket s' \rrbracket$) and this is easily seen to be equivariant too, since $\mathcal{C}$ has equivariant structure.
- An equivariant map $\llbracket - \rrbracket : Fun_{Sg} \longrightarrow ob\,\mathcal{C}$ where for each higher order function constant $c : s$ we have $\llbracket c \rrbracket : 1 \longrightarrow \llbracket s \rrbracket$ (recall that $\mathcal{C}$ has finite products—hence an equivariant terminal object).

Let $\nabla = \overline{a}_1 \# x_1 : s_1, ..., \overline{a}_n \# x_n : s_n \in Env_{Sg}$ be a freshness environment. Then we define the $\mathcal{C}$-object $\llbracket \nabla \rrbracket$ by $\llbracket \nabla \rrbracket \overset{\text{def}}{=} \llbracket s_1 \rrbracket^{\# \overline{a}_1} \times ... \times \llbracket s_n \rrbracket^{\# \overline{a}_n}$. We define a notion of satisfaction for both expressions-in-context and equations-in-context. Let $\mathcal{M}$ be a structure for a NLC-signature in an FM-ccc $\mathcal{C}$ and consider the binary relation $\blacktriangleright$ in Table 3. Table 3 specifies a partial function $J \mapsto \llbracket J \rrbracket$ from judgements to morphisms $\llbracket J \rrbracket$ in $\mathcal{C}$. Given $\nabla \vdash^E M : s$ or $\nabla \vdash^E \overline{a} \# M : s$ we say that $\mathcal{M}$ **satisfies** the judgement if the morphism $\llbracket \nabla \vdash^E M : s \rrbracket : \llbracket \nabla \rrbracket \longrightarrow \llbracket s \rrbracket$ or $\llbracket \nabla \vdash^E \overline{a} \# M : s \rrbracket : \llbracket \nabla \rrbracket \longrightarrow \llbracket s \rrbracket^{\# \overline{a}}$ in $\mathcal{C}$ is defined (that is, the partial function $J \mapsto \llbracket J \rrbracket$ is defined). If so we write $\llbracket \nabla \vdash^E M : s \rrbracket \Downarrow$ or $\llbracket \nabla \vdash^E \overline{a} \# M : s \rrbracket \Downarrow$. Generally, $\llbracket J \rrbracket \Downarrow \overset{\text{def}}{=} (\exists j)(\llbracket J \rrbracket \blacktriangleright j)$. We may write $\llbracket J \rrbracket$ or even $\llbracket J \rrbracket \Downarrow$ for morphism j. Given $\nabla \vdash^E M \approx M' : s$ we say that $\mathcal{M}$ **satisfies** it if both $\llbracket \nabla \vdash^E M : s \rrbracket \Downarrow$ and $\llbracket \nabla \vdash^E M' : s \rrbracket \Downarrow$ and they are equal morphisms in $\mathcal{C}$. We say that $\mathcal{M}$ is a **model** of a NLC theory $Th = (Sg, Ax)$ if $\mathcal{M}$ satisfies all of the equations-in-context in Ax. With this, we have our soundness theorem:

Theorem 4.2 (Soundness) *Let Th be a* NLC *theory and $\mathcal{M}$ a model of Th in an FM-ccc. Then every typed expression $Th \rhd \nabla \vdash^E M : s$, freshness assertion $Th \rhd \nabla \vdash^E \overline{a} \# M : s$ and theorem $Th \rhd \nabla \vdash^E M \approx M' : s$ is satisfied by $\mathcal{M}$.*

We need the following intermediate results to prove the soundness theorem. We adopt a direct approach to proving that our semantics is compositional with respect to substitution, which reduces some overhead from the approach in [7]. Note that we appeal to Propositions 3.4 and Proposition 3.5 to ensure that the NLC judgements mentioned below are properly defined. We shall write $L \asymp R$ to mean that $L\Downarrow \iff R\Downarrow$ and that $L = R$. An example proof for part of Lemma 4.4 is in the Appendix.

Lemma 4.3 (Semantic Id, Inclusion, Int. Perm. Action, Projection) *Given a freshness environment $\nabla = \overline{a}_1 \# x_1 : s_1, ..., \overline{a}_n \# x_n : s_n$ then we have*

(i) $id_{[\![\nabla]\!]} \asymp \langle [\![\nabla \vdash \overline{a_1} \# x_1 : s_1]\!], ..., [\![\nabla \vdash \overline{a_n} \# x_n : s_n]\!]\rangle$

(ii) $i^{\overline{a}}_{[\![\nabla]\!]} \asymp \langle [\![\nabla^{\#\overline{a}} \vdash \overline{a_1} \# x_1 : s_1]\!], ..., [\![\nabla^{\#\overline{a}} \vdash \overline{a_n} \# x_n : s_n]\!]\rangle$

(iii) $\pi_{[\![\nabla]\!]} \asymp \langle [\![\nabla \vdash \pi \cdot \overline{a_1} \# \pi x_1 : \pi \cdot s_1]\!], ..., [\![\nabla \vdash \pi \cdot \overline{a_n} \# \pi x_n : \pi \cdot s_n]\!]\rangle$

(iv) $pr_{[\![\nabla_j]\!]} : [\![\nabla_1]\!] \times [\![\nabla_2]\!] \to [\![\nabla_j]\!] \asymp \langle [\![\boldsymbol{\nabla_1 \cup \nabla_2 \vdash \overline{a_i} \# x_i : s_i}]\!]\rangle$*, where* $\nabla_1, \nabla_2 \in Env_{Sg}$ *have disjoint domains but are such that* $\nabla_j = \nabla$ *for* $j = 1$ *and* 2.

Lemma 4.4 (Useful Semantic Factorisations "$[\![\xi]\!] = [\![\xi]\!] \circ m$")

(i) *The function* $[\![-]\!] : Env_{Sg} \to Env_{Sg}$ *is equivariant.*

(ii) $[\![\pi \cdot \nabla \vdash^E \pi \cdot M : \pi \cdot s]\!] \asymp \pi \cdot [\![\nabla \vdash^E M : s]\!]$

(iii) $[\![\nabla \vdash \pi * M : \pi \cdot s]\!] \asymp \pi_{[\![s]\!]} \circ [\![\nabla \vdash M : s]\!]$

(iv)
$$[\![\nabla, \pi \cdot \overline{a} \# x : \pi \cdot s \vdash^E M\{\pi^{-1}/x\} : s']\!] \asymp [\![\nabla, \overline{a} \# x : s \vdash^E M : s']\!] \circ (id_{[\![\nabla]\!]} \times \pi^{-1}_{[\![\pi \cdot s]\!]^{\#\pi \cdot \overline{a}}})$$

(v) *Given $\nabla \leq \nabla'$ there exists an arrow weak : $[\![\nabla']\!] \to [\![\nabla]\!]$ such that for any typed expression $\nabla \vdash^E M : s$, $[\![\nabla' \vdash^E M : s]\!] \asymp [\![\nabla \vdash^E M : s]\!] \circ weak$.*

(vi) $[\![\nabla^{\#\overline{a}} \vdash^E M : s]\!] \asymp [\![\nabla \vdash^E M : s]\!] \circ i^{\overline{a}}$ *where* $\overline{a} \# \nabla$.

Proposition 4.5 (Compositional Semantics) *Let $\nabla \overset{\text{def}}{=} \overline{a}_1 \# x_1 : s_1, ..., \overline{a}_n \# x_n : s_n$. Suppose, for theory Th, we have the typed expression $\nabla \vdash^E M : s$ and freshness assertions $\nabla' \vdash^E \overline{a}_i \# N_i : s_i$ for each i. Then we have $\nabla' \vdash^E M\{\boldsymbol{N_i/x_i}\} : s$. Moreover, if $[\![\nabla \vdash^E M : s]\!]\Downarrow$ and $[\![\nabla' \vdash^E \overline{a}_i \# N_i : s_i]\!]\Downarrow$ for each i then we have $[\![\nabla' \vdash^E M\{\boldsymbol{N_i/x_i}\} : s]\!]\Downarrow$ and further $[\![\nabla' \vdash^E M\{\boldsymbol{N_i/x_i}\} : s]\!] = [\![\nabla \vdash^E M : s]\!] \circ \langle [\![\boldsymbol{\nabla' \vdash^E \overline{a}_i \# N_i : s_i}]\!]\rangle$.*

The proofs of Lemmas 4.3 and 4.4 require a combination of direct calculations and inductions over the structure of terms. Note that the proof of Proposition 4.5 is by induction over the structure of M and does not require a complicated statement that is provable by mutual induction. The intuition is that, as one would expect, the semantics of expressions is derivation independent. We are now in a position to prove Theorem 4.2—see Appendix page 109.

5 A Complete Categorical Semantics

In order to obtain a completeness result we need a way in which we can construct a cartesian closed category out of NLC syntax. To do this we augment the types, expressions and rules with a form of atom-abstraction. In doing so we arrive at the final form of NLC (with abstraction) for which we have a categorical model that is both sound and complete. Please note that we only give a summary of the details in this preliminary paper.

We augment the type system with types of the form $\text{И}a.\, s$. We augment the collection of terms with abstraction and concretion terms $\langle a \rangle\, M$ and $M @ a$. The permutation actions on the resulting expressions are defined in the expected way. The type system and equation system appears on page 104. The equations specify forms of beta-rule and eta-rule, ensure that term forming operations are congruences, and that the $\langle a \rangle\, M$ abstraction operator on expressions is equated with $\langle a' \rangle\, M'$ provided that the two expressions given by swapping out the a and a' for a fresh atom b are provable equal in the logic.

We also need a richer categorical structure to achieve completeness. For any FM-category, there is a family of categories $(\mathcal{C}^{\#a} \mid a \in \mathbb{A})$ where $ob\, \mathcal{C}^{\#a}$ consists of those $C \in ob\, \mathcal{C}$ for which $a \# \mathcal{C}$. Given such $C, C' \in \mathcal{C}^{\#a}$, then $f : C \to C' \in mor\, \mathcal{C}$ is a morphism in $\mathcal{C}^{\#a}$ just in case $a \# f$. The basic properties of fresh inclusions ensure that each $\mathcal{C}^{\#a}$ is indeed a category, and moreover that there is a functor $(-)^{\#a} : \mathcal{C}^{\#a} \to \mathcal{C}$. We shall require this functor to have a right adjoint $\text{И}a\,.(-)$ and for there to be a family of morphisms $conc_b : (\text{И}a\,.C)^{\#b} \to (a\, b) \cdot C$. These structures are required to satisfy commutativity properties which are needed in order to soundly model the equations $\textsf{BAA}$ and $\textsf{EAA}$ (see Figure A.3). For example, for every $D \in ob\, \mathcal{C}^{\#a}$, $X \in ob\, \mathcal{C}$, and $a', b \# X$, where $\eta_{a,D}$ is the counit of the adjunction, we have

$$\begin{tikzcd}
D^{\#a} \arrow[r, "m"] \arrow[d, "i"'] & (a\, a') \cdot X \arrow[r, "(a'\, b)_{(a\, a') \cdot X}"] & (a\, b) \cdot X \\
D \arrow[rr, "F^*"'] & & (\text{И}a\,.X)^{\#b} \arrow[u, "conc_b"']
\end{tikzcd}$$

with F being the morphism

$$\begin{tikzcd}
D \arrow[r, "\eta_{a,D}"] & \text{И}a\,.D^{\#a} \arrow[r, "\text{И}a\,.m"] & \text{И}a\,.(a\, a') \cdot X \arrow[r, "\text{И}a\,.(a\, a')_{(a\, a') \cdot X}"] & \text{И}a\,.X
\end{tikzcd}$$

Further

$$\begin{tikzcd}
D \arrow[r, "F"] \arrow[d, "\eta_{a,D}"'] & \text{И}a\,.X \arrow[d, equal] \\
\text{И}a\,.D^{\#a} \arrow[r, "\text{И}a\,.((a\, b)_{(a\, b) \cdot X} \circ conc_a \circ F^* \circ i^a_D)"'] & \text{И}a\,.X
\end{tikzcd}$$

$$\frac{[\![\nabla^{\#a} \vdash^E M : (a\,a')\cdot s]\!] \blacktriangleright m : [\![\nabla]\!]^{\#a} \to (a\,a')\cdot[\![s]\!]}{[\![\nabla \vdash^E \langle a'\rangle\, M : \mathsf{N}a.\,s]\!] \blacktriangleright \mathsf{N}a\,.((a\,a')_{(a\,a')\cdot[\![s]\!]} \circ m) \circ \eta_{a,[\![\nabla]\!]} : [\![\nabla]\!] \to \mathsf{N}a\,.[\![\nabla]\!]^{\#a} \to \mathsf{N}a\,.[\![s]\!]}$$

$$\frac{[\![\nabla \vdash^E F : \mathsf{N}a.\,s]\!] \blacktriangleright f : [\![\nabla]\!] \to \mathsf{N}a\,.[\![s]\!]}{[\![\nabla^{\#a} \vdash^E F @ b : (a\,b)\cdot s]\!] \blacktriangleright conc_b \circ f^* \circ i^a_{[\![\nabla]\!]} : [\![\nabla]\!]^{\#a} \to [\![\nabla]\!] \to (\mathsf{N}a\,.[\![s]\!])^{\#b} \to (a\,b)\cdot[\![s]\!]}\ \dagger(f,b)$$

Table 3
Semantics in an FM-ccc

where

$$\begin{tikzcd} D^{\#a} \arrow[r, "i^a_D"] & D \arrow[r, "F^*"] & (\mathsf{N}a\,.X)^{\#b} \arrow[r, "conc_b"] & (a\,b)\cdot X \arrow[r, "(a\,b)_{(a\,b)\cdot X}"] & X \end{tikzcd}$$

Suppose that we also require the adjoints to commute. We call such FM-cccs with this additional structure NFM-cccs; it is these categories that yield a sound and complete semantics for NLC.

An example of such a category is *FMSet*. The action of the functor $(-)^{\#a}$ sends any FM-function $f : X \to Y \in \mathit{FMSet}^{\#a}$ to $f^{\#a} : X^{\#a} \to Y^{\#a}$ where $f^{\#a}(x \in X^{\#a}) \stackrel{\text{def}}{=} f(x) \in Y^{\#a}$ is easily seen to be well-defined. The action of the right adjoint $\mathsf{N}a\,.(-)$ is defined on $f : X \to Y$ by setting

$$\mathsf{N}a\,.X \stackrel{\text{def}}{=} \{\langle a'\rangle\, x \mid a' \# X \wedge x \in (a\,a')\cdot X\}$$

where $\langle a'\rangle\, x$ is the abstraction operator of Gabbay and Pitts [14], and $\mathsf{N}a\,.f(z \in \mathsf{N}a\,.X) \stackrel{\text{def}}{=} ((a\,b)\cdot f)(z @ b)$ for some/any suitably fresh atom b. The verification that we have an adjunction satisfying the stated properties is a rather length calculation which we omit from this paper.

The Classifying Category and Categorical Completeness. The notion of classifying category, topos, etc is a standard one in category theory [10,22]. To prove completeness we now show that we can build an FM-ccc from the syntax of a NLC theory (Proposition 5.1), together with a generic model [10] (Propositions 5.3 and 5.4). Sketch proofs are in the Appendix.

Proposition 5.1 (Classifying Category) *For every* NLC*-theory Th we can define a classifying FM-ccc $Cl(Th)$ which is built from the syntax of Th. An object is a freshness environment $\nabla \stackrel{\text{def}}{=} (\overline{a}_1 \# x_1 : s_1, ..., \overline{a}_n \# x_n : s_n)$. If $\nabla' \stackrel{\text{def}}{=} (\overline{a'}_1 \# x'_1 : s'_1, ..., \overline{a'}_m \# x'_m : s'_m)$ then a morphism $\delta \stackrel{\text{def}}{=} ([M_1]_\approx, \ldots, [M_m]_\approx) : \nabla \to \nabla'$ is a list of typed expressions such that for $1 \leq i \leq m$ we have $Th \rhd \nabla \vdash^E \overline{a}'_i \# M_i : s'_i$, and $[M_i]_\approx$ is the equivalence class of those T such that $Th \rhd \nabla \vdash^E M_i \approx T : s'_i$.*

Remark 5.2 We explain, with a simple example, how we are able to construct exponentials in the classifier. Consider $(a_1 \# x_1 : s_1) \Rightarrow (a'_1 \# x'_1 : s'_1)$. One would imagine that, whatever the exponential is, it should somehow involve the type ${s_1}^{a_1} \Rightarrow {s'_1}^{a'_1}$ which is not legitimate in NLC. However, consider the following, recalling that in an ИFM-ccc the adjoints commute

$$C_1^{\#a_1} \Rightarrow C_1'^{\#a_1'} \cong (\text{И}a_1' .C_1^{\#a_1} \Rightarrow C_1')^{\#a_1'} \cong (\underbrace{(\text{И}a_1' .C_1)^{\#a_1} \Rightarrow C_1'}_{\text{"valid NLC type"}})^{\#\boxed{a_1'}}$$

We mimic the above isomorphisms in the syntax of NLC in order to construct exponentials, giving brief details in the appendix. The freshness assertion $\boxed{a_1'}$ is captured by an NLC freshness assertion.

Proposition 5.3 *The **generic** Sg-structure $\mathcal{G}$ of $Th = (Sg, Ax)$ in $Cl(Th)$ is given by defining $[\![\gamma]\!]_{\mathcal{G}} =_{def} (\emptyset \# x : s)$ where γ is any ground type from Sg. If c is a constant with typing $c : s$ then $[\![c]\!]_{\mathcal{G}} \overset{\text{def}}{=} ([c]_{\approx}) : 1 \overset{\text{def}}{=} () \longrightarrow (\emptyset \# x : s)$ is well defined since $Th \rhd [\,] \vdash^E c : s$ Further, suppose that $Th \rhd \nabla \vdash^E M : s$. Then $[\![\nabla \vdash^E M : s]\!]_{\mathcal{G}} \blacktriangleright [M]_{\approx} : \nabla \to (\emptyset \# v : s)$*

Proposition 5.4 *The generic structure $\mathcal{G}$ is a model of any $Th = (Sg, Ax)$.*

Theorem 5.5 (Completeness) *The categorical semantics of* NLC*-theories in FM-cccs is complete: Let Th be a* NLC*-theory. If any equation-in-context for Th is satisfied in all FM-ccc models of Th, then it is a theorem.*

6 Category Theory/Type Theory Correspondence

Clouston [7] demonstrated a categorical type theory correspondence between NEL and FM-categories; we have established a similar correspondence between NLC and FM-cccs. Recall [10] that the correspondence result for standard λ-calculus and cartesian closed categories is slightly more restricted than the one for EL and categories with finite products: Due to the covariant nature of exponentials, components of homomorphisms of models must be restricted to isomorphisms.

Theorem 6.1 *The category $Cl(Th)$ is a **classifying category** for* NLC*-theories in the sense that for every model $\mathcal{M}$ of Th in a ИFM-ccc $\mathcal{D}$ there is a unique ИFM-ccc functor $F_{\mathcal{M}} : Cl(Th) \to \mathcal{D}$ such that $F_{\mathcal{M}}$ composes with the generic model to yield $\mathcal{M}$.*

Now take a definition of homomorphism $h : \mathcal{M} \to \mathcal{N}$ of models of an NLC-theory Th in an ИFM-ccc $\mathcal{C}$ consisting of **equivariant** isomorphisms $h_\gamma : [\![\gamma]\!]_{\mathcal{M}} \cong [\![\gamma]\!]_{\mathcal{M}}$, where $h_{s^{\overline{a}} \Rightarrow s'}$ is given by $(h_s^{\#\overline{a}})^{-1} \Rightarrow h_{s'}$ (and $\overline{a} \# s$ ensures homomorphisms are well defined). For a NLC-theory Th and a small ИFM-ccc $\mathcal{C}$, the **category of models** $\mathcal{M}od_{\cong}(Th, \mathcal{C})$ consists of the Th models and homomorphisms. An ИFM-ccc functor $F : \mathcal{C} \to \mathcal{D}$ is an ИFM-functor that preserves exponentials and commutes with the adjunction. We can define $FMccc_{\cong}(\mathcal{C}, \mathcal{D})$ as a category with ИFM-ccc functors as objects and finitely supported natural isomorphisms as morphisms.

Theorem 6.2 *We have $FMccc_{\cong}(Cl(Th), \mathcal{D}) \simeq \mathcal{M}od_{\cong}(Th, \mathcal{D})$ for any* NLC*-theory Th and ИFM-ccc $\mathcal{D}$. For any ИFM-ccc $\mathcal{C}$, we have $Cl(Th(\mathcal{C})) \simeq \mathcal{C}$. For any* NLC*-theory Th we have $Th \simeq Th(Cl(Th))$.*

7 Solutions, Open Questions, and Further Work

Exploiting Atom-Dependent Types. Clouston [4] observes that name-abstraction and concretion in *FMSet* cannot be captured by a NEL theory. This is related to the fact that concretion is a partial function, which can only be applied to arguments that meet certain freshness conditions. In the total concretion theory in Section 8 of [4] (page 15; MFPS 2010), Clouston describes concretion functions of the form $con_a : Name.s \to s$ where $Name.s$ is the name-abstraction type. Now $con_a\, x$ is well-formed only if $a \# x$. NEL does not support such partiality. But in NLC we have exploited the new dependent type system to yield a solution. This provides us with an alternative way to handle concretion, without relying on local scoping [29] or bunched contexts [3]. We intend to fully develop this in an extended journal article.

Internal and 2-Categorical Approaches Could this paper be simplified by considering internalisation in one of the FM-toposes? We cannot give a definitive answer: a deep investigation must wait for future work, but here are a few observations. Consider even the basic notion of perm-category. A perm-category is internal to *FMNom*; but *an internal FMNom-category is not a perm-category* since the morphism permutation π_C is not directly captured by the internalisation. So it is not clear to us that the notion of FM-ccc could be extracted by internalisation. Going further, it is also not clear how the atom-set-partiality of our higher types can be (usefully) captured. However, even if it can, this misses a central point of our paper: a direct investigation into the interplay of higher order types and the freshness relation via a domain specific formal type theory. Possibly if one sought a direct route to "some kind of" completeness result an internal approach might work, but we are trying to do more than that. What is true is that the "nominal" world still needs to be better understood from a "2-categorical" viewpoint, and there are a number of open questions.

Future Work. Recall our motivation for this work: to develop a formal framework for nominal higher order functions, with a view to proving it a conservative extension over NEL by nominal gluing. Nominal gluing remains work in progress, but our preliminary results about the Yoneda Lemma and cartesian closure of nominal functor categories appear in [12]. From such a gluing proof, we might be able to extract a form of categorical normalisation result, taking the work in a more applied direction through the construction of some form of abstract machine for NLC along with an implementation. Is there some form of nominal categorical abstract machine?

From the Computer Science perspective, we have taken great care in specifying NLC formally and care with proofs that involve quite subtle intricacies arising from α-equivalence in the nominal setting, and the (variable) equivariance of judgements and rules. We have attempted to avoid the traps (explained in [28]) that others have fallen into. As such, it would be an interesting project to study a mechanisation of NLC.

How much further can one take categorical correspondences for nominal logics/type theories? We are considering product and coproduct types, and of course one might study computational monad types, numbers, and more [20]. Going still further there is the general consideration of Martin Löf dependent type theory [27,32], nominal and FM analogues, and corresponding categorical structures. We are also investigating Henkin style models as have Gabbay and Mulligan [17]. Cheney [3] has studied the properties of a type theory that mixes functions, and atoms as first class citizens, along with a name abstraction operator. While discussing others' work, it is interesting to note that *type dependency* is a common feature of studies involving computational type-and-effect systems. Examples are [34,1].

We have considered the possibility that the original NEL could be presented using dependent types in place of freshness assertions. However, the resulting type theory might be different. Such dependently typed theories, in which $\overline{a} \# x : s$ is wholly replaced by $x : s^{\overline{a}}$, could be more expressive than NEL theories. This remains future work.

We thank Ranald Clouston for very detailed comments; John Power for being helpful and generous with his time over a public holiday; and for other useful comments and observations from Martin Hyland and Bill Lawvere. We must also thank others who have provided useful thoughts that we hope have improved this paper. Finally, and most importantly, we thank Andrew Pitts for extensive discussions about an earlier version of this paper which contained an error and assisting with its correction.

References

[1] Nick Benton, Andrew Kennedy, Lennart Beringer, and Martin Hofmann. Relational Semantics for Effect-Based Program Transformations with Dynamic Allocation. In *Proc. of the 9th ACM SIGPLAN international conference on Principles and Practice of Declarative Programming*, PPDP '07, pages 87–96, New York, NY, USA, 2007. ACM.

[2] Clemens Berger, Paul-Andre Mellies, and Mark Weber. Monads with Arities and their Associated Theories. *Journal of Pure and Applied Algebra*, 216(89):2029–2048, 2012.

[3] James Cheney. A Dependent Nominal Type Theory. *Logical Methods in Computer Science*, 8(1), 2012.

[4] Ranald Clouston. Binding in Nominal Equational Logic. *Electr. Notes Theor. Comput. Sci.*, 265:259–276, 2010.

[5] Ranald Clouston. Nominal Lawvere Theories. In *WoLLIC'11*, pages 67–83, 2011.

[6] Ranald Clouston. Nominal Logic with Equations Only. In *Logical Frameworks, Metalanguages and Theory of Programming*, pages 44–57, 2011.

[7] Ranald Clouston. Nominal Lawvere Theories: A Category Theoretic Account of Equational Theories with Names. *Journal of Computer and System Sciences*, 2013.

[8] Ranald Clouston and Andrew M. Pitts. Nominal Equational Logic. *Electron. Notes Theor. Comput. Sci.*, 172:223–257, 2007.

[9] R. L. Crole. On Fixpoint Objects and Gluing Constructions. *Applied Categorical Structures*, 4(2 & 3):251–281, 1996. This volume is a Special Edition for the European Colloquium on Category Theory, Tours, France.

[10] Roy L. Crole. *Categories for Types.* Cambridge University Press, 1993.

[11] Roy L. Crole. α-Equivalence Equalities. *Theoretical Computer Science*, 433:1–19, May 2012.

[12] Roy L. Crole and Frank Nebel. The Yoneda Lemma and Cartesian Closure in the FM-World. Submitted, 2013.

[13] P.J. Freyd and A. Scedrov. *Categories, Allegories.* Elsevier Science Publishers, 1990. Appears as Volume 39 of the North-Holland Mathematical Library.

[14] Murdoch Gabbay and Andrew M. Pitts. A New Approach to Abstract Syntax with Variable Binding. *Formal Asp. Comput.*, 13(3-5):341–363, 2002.

[15] Murdoch J. Gabbay. Foundations of Nominal Techniques: Logic and Semantics of Variables in Abstract Syntax. *Bulletin of Symbolic Logic*, 17(2):161–229, 2011.

[16] Murdoch J. Gabbay and Aad Mathijssen. Nominal Universal Algebra: Equational Logic with Names and Binding. *Journal of Logic and Computation*, 19(6):1455–1508, December 2009.

[17] Murdoch James Gabbay and Dominic P. Mulligan. Nominal Henkin Semantics: Simply-Typed Lambda-Calculus Models in Nominal Sets. In *LFMTP*, pages 58–75, 2011.

[18] J.R. Hindley and J.P. Seldin. *Introduction to Combinators and the Lambda Calculus*, volume 1 of *London Mathematical Society Student Texts*. Cambridge University Press, 1988.

[19] Martin Hyland and John Power. The Category Theoretic Understanding of Universal Algebra: Lawvere Theories and Monads. *Electr. Notes Theor. Comput. Sci.*, 172:437–458, 2007.

[20] Neelakantan R. Krishnaswami and Nick Benton. Adding Equations to System F Types. In *ESOP*, pages 417–435, 2012.

[21] J. Lambek. From λ-calculus to cartesian closed categories. In J.P. Seldin and J.R. Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*. Academic Press, 1980.

[22] J. Lambek and P.J. Scott. *Introduction to Higher Order Categorical Logic*. Cambridge University Press, 1986.

[23] F.W. Lawvere. *Functorial Semantics of Algebraic Theories*. PhD thesis, Columbia University, 1963. Summary appears in Proc. of the National Academy of Science, 50:869–873, 1963.

[24] E. Manes. *Algebraic Theories*, volume 26 of *Graduate Texts in Mathematics*. SV, 1976.

[25] James McKinna and Robert Pollack. Some Lambda Calculus and Type Theory Formalized. *JAR*, 1998.

[26] Paul-André Melliès. Segal Condition Meets Computational Effects. In *LICS*, pages 150–159, 2010.

[27] B. Nordström, K. Petersson, and J.M. Smith. *Programming in Martin-Löf's Type Theory*, volume 7 of *Monographs on Computer Science*. Oxford University Press, 1990.

[28] A. M. Pitts. Nominal Logic, A First Order Theory of Names and Binding. *Information and Computation*, 186:165–193, 2003.

[29] A. M. Pitts. Structural Recursion with Locally Scoped Names. *Journal of Functional Programming*, 21(3):235–286, 2011.

[30] A. M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*, volume 57 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2013.

[31] Andrew M. Pitts. Categorical Logic. In *Handbook of Logic in Computer Science: Volume 5: Logic and Algebraic Methods*, pages 39–123, Oxford, UK, 2000. Oxford University Press.

[32] Thomas Streicher. Independence Results for Calculi of Dependent Types. In *Category Theory and Computer Science*, pages 141–154, 1989.

[33] Thomas Streicher. *Semantics of Type Theory: Correctness, Completeness, and Independence Results*, volume XII of *Progress in Theoretical Computer Science*. Basel: Birkhäuser Verlag, 1991.

[34] Jacob Thamsborg and Lars Birkedal. A Kripke Logical Relation for Effect-Based Program Transformations. In *Proc of the 16th ACM SIGPLAN international conference on functional programming*, ICFP '11, pages 445–456, New York, NY, USA, 2011. ACM.

A Inductive Definition of Type System and Equations

$$\textsf{(SP)}\quad \frac{}{\nabla, \overline{a} \mathrel{\#} x : s \vdash^E \pi x : \pi \cdot s}$$

$$\textsf{(C)}\quad \frac{}{\nabla \vdash^E c : s}\ (c \in \mathit{Fun}_{\mathit{Sg}} \text{ and } c \text{ has } \mathit{Sg} \text{ typing } c : s)$$

$$\textsf{(ABS)}\quad \frac{\nabla, \overline{a} \mathrel{\#} x : s \vdash^E M : s'}{\nabla \vdash^E \lambda^{\overline{a}} x : s.\, M : s^{\overline{a}} \Rightarrow s'}$$

$$\textsf{(AP)}\quad \frac{\nabla \vdash^E F : s^{\overline{a}} \Rightarrow s' \qquad \nabla \vdash^E \overline{a} \mathrel{\#} A : s}{\nabla \vdash^E F\ A : s'}$$

$$\textsf{(AE*)}\quad \frac{\nabla^{\# \overline{a}} \vdash^E M : s}{\nabla \vdash^E M : s}\ (\overline{a} \mathrel{\#} (\nabla, M))$$

$$\textsf{(WEAK*)}\quad \frac{\nabla \vdash^E M : s}{\nabla' \vdash^E M : s}\ (\nabla \leq \nabla')$$

$$\textsf{(SUB*)}\quad \frac{\nabla' \vdash^E \overline{a_i} \mathrel{\#} N_i : s_i \qquad \nabla \vdash^E M : s'}{\nabla' \vdash^E M\{N_1, \ldots, N_n / x_1, \ldots, x_n\} : s'}$$

In rule (SUB*) $\quad \nabla \overset{\text{def}}{=} \overline{a}_1 \mathrel{\#} x_1 : s_1, ..., \overline{a}_n \mathrel{\#} x_n : s_n$ and $1 \leq i \leq n$

Table A.1
NLC Typing Rules for a Given *Th*

Note that the rules marked by an asterisk are provably admissible.

B A Results Road Map

The structured requirements list below shows the dependencies of results in the paper. Any result depends on those that appear at deeper nestings. This SRL is for guidance only!

- Categorical type theory correspondence Theorem 6.2 and classifying category Theorem 6.1
  - · Completeness Theorem 5.5
    - Existence of generic categorical model Proposition 5.4.
    - Generic model (of *Th* in *Cl*(*Th*)) properties Proposition 5.3
    - Building a classifying category *Cl*(*Th*) Proposition 5.1
      - ⇐ PROPOSITIONS
  - · Soundness Theorem 4.2
    - Semantics is compositional Proposition 4.5

$$(\mathsf{REF})\ \frac{\nabla \vdash^E M : s}{\nabla \vdash^E M \approx M : s} \qquad (\mathsf{SYM})\ \frac{\nabla \vdash^E M \approx M' : s}{\nabla \vdash^E M' \approx M : s}$$

$$(\mathsf{TRANS})\ \frac{\nabla \vdash^E M \approx M' : s \qquad \nabla \vdash^E M' \approx M'' : s}{\nabla \vdash^E M \approx M'' : s}$$

$$(\mathsf{WEAK})\ \frac{\nabla \vdash^E M \approx M' : s}{\nabla' \vdash^E M \approx M' : s}\ (\nabla \leq \nabla')$$

$$(\mathsf{AE})\ \frac{\nabla^{\#\overline{a}} \vdash^E M \approx M' : s}{\nabla \vdash^E M \approx M' : s}\ (\overline{a} \# (\nabla, M, M'))$$

$$(\mathsf{PERM})\ \frac{\nabla \vdash^E M : s}{\nabla^{\#ds(\pi,\pi')} \vdash^E \pi * M \approx \pi' * M : \pi \cdot s}\ (ds(\pi, \pi') \# (\nabla, M))$$

$$(\mathsf{BF})\ \frac{\nabla, \overline{a} \# x : s \vdash^E M : s' \qquad \nabla \vdash^E \overline{a} \# N : s}{\nabla \vdash^E (\lambda^{\overline{a}} x : s.\, M)\ N \approx M\{N/x\} : s'}$$

$$(\mathsf{EF})\ \frac{\nabla \vdash^E M : s^{\overline{a}} \Rightarrow s'}{\nabla \vdash^E \lambda^{\overline{a}} x : s.\, (M\ x) \approx M : s^{\overline{a}} \Rightarrow s'}\ (x \notin fv(M))$$

$$(\mathsf{CL})\ \frac{\nabla, \overline{a} \# x : s \vdash^E M \approx M' : s'}{\nabla \vdash^E \lambda^{\overline{a}} x : s.\, M \approx \lambda^{\overline{a}} x : s.\, M' : s^{\overline{a}} \Rightarrow s'}$$

$$(\mathsf{CA})\ \frac{\nabla \vdash^E \overline{a} \# A_i : s \qquad \nabla \vdash^E F_1 \approx F_2 : s^{\overline{a}} \Rightarrow s' \qquad \nabla \vdash^E A_1 \approx A_2 : s}{\nabla \vdash^E F_1\ A_1 \approx F_2\ A_2 : s'}\ \text{(i=1,2)}$$

$$(\mathsf{SUB})\ \frac{\begin{array}{l} \nabla' \vdash^E \overline{a_i} \# N'_i : s_i \\ \nabla' \vdash^E \overline{a_i} \# N_i : s_i \qquad \nabla' \vdash^E N_i \approx N'_i : s_i \qquad \nabla \vdash^E M \approx M' : s' \end{array}}{\nabla' \vdash^E M\{N_1, \ldots, N_n/x_1, \ldots, x_n\} \approx M'\{N'_1, \ldots, N'_n/x_1, \ldots, x_n\} : s'}$$

$ds(\pi, \pi')$ is the **disagreement set**: $\{a \in \mathbb{A} \mid \pi(a) \neq \pi(a)\}$

In rule (SUB) $\quad \nabla \overset{\text{def}}{=} \overline{a}_1 \# x_1 : s_1, ..., \overline{a}_n \# x_n : s_n$ and $1 \leq i \leq n$

Table A.2
NLC Equation Rules for a Given *Th*

Properties of semantics: factorisation results Lemma 4.4
Properties of semantics: denotations of syntactic *id*, *i*, π Lemma 4.3
Properties of FM-cccs Lemma D.1
⇐ PROPOSITIONS

- PROPOSITIONS:

$$(\textsf{AABS})\quad \frac{\nabla^{\# a} \vdash^{E} M : (a\, a') \cdot s}{\nabla \vdash^{E} \langle a' \rangle\, M : \text{И} a.\, s}\ (a \# \nabla, M, a' \# s)$$

$$(\textsf{CONC})\quad \frac{\nabla \vdash^{E} b \# F : \text{И} a.\, s}{\nabla^{\# a} \vdash^{E} F \mathbin{@} b : (a\, b) \cdot s}\ (a \# \nabla, b \# F)$$

$$(\textsf{BAA})\quad \frac{\nabla^{\# a} \vdash^{E} M : (a\, a') \cdot s}{\nabla^{\# a} \vdash^{E} \langle a' \rangle\, M \mathbin{@} b \approx (a'\, b) \cdot M : (a\, b) \cdot s}\ (a \# \nabla, M, a' \# s)$$

$$(\textsf{EAA})\quad \frac{\nabla \vdash^{E} b \# F : \text{И} a.\, s}{\nabla^{\# a} \vdash^{E} \langle b \rangle\, (F \mathbin{@} b) \approx F : (a\, b) \cdot s}\ (a \# \nabla, b \# F)$$

$$(\textsf{CAA})\quad \frac{\nabla^{\# a} \vdash^{E} M \approx M' : (a\, a') \cdot s}{\nabla \vdash^{E} \langle a' \rangle\, M \approx \langle a' \rangle\, M' : \text{И} a.\, s}\ (a \# \nabla, M, a' \# s)$$

$$(\textsf{BINDAA})\quad \frac{\nabla^{\# a, z} \vdash^{E} (z\, a') \cdot M \approx (z\, a'') \cdot M' : (a\, a') \cdot s}{\nabla \vdash^{E} \langle a' \rangle\, M \approx \langle a'' \rangle\, M' : \text{И} a.\, s}\ (a \# \nabla, M, a', a'' \# s)$$

$$(\textsf{CC})\quad \frac{\nabla \vdash^{E} b \# F : \text{И} a.\, s \qquad \nabla \vdash^{E} F \approx F' : \text{И} a.\, s}{\nabla^{\# a} \vdash^{E} F \mathbin{@} b \approx F' \mathbin{@} b : (a\, b) \cdot s}\ (a \# \nabla, b \# F, F')$$

Table A.3
NLC Augmented Typing and Equation Rules for a Given *Th*

· Typed expressions and theorems are equivariant Proposition 3.5 and the object level permutation action preserves equalities Proposition 3.4

Permutation actions on freshness hypotheses iff suspension substitutions on expressions Lemma 3.3

NLC is equvariant under variable swapping Lemma 3.2

Meta-level perm action distributes over substitution Proposition 2.5

Object-level perm action associates with substitution Proposition2.4

Capture avoiding substitution is well defined Lemma C.4

Object-level permutation action in terms of meta-level permutation action Lemma C.3

Permutation actions are well defined Proposition 2.2

◇ Suspension substitution is ordinary substitution Lemma C.2

◇ Meta-level permutation action associates with suspension substitutions Lemma C.1

Please note that in our main text, by making use of a blue colour we indicate "signposts" giving "directions" through the paper.

We make occasional emphasis with a green colour.

C Substitution and α-Equivalence

In this section of the Appendix we explain some of the details of substitution, α-equivalence and the associated lemmas that underpin our results. Note that we believe our approach, based on untyped terms, unifies and simplifies the results in [7]. The lemmas are proved by induction; one example is in Section E.

Lemma C.1 *$(\pi * M)[\pi^{-1}x/x] = \pi * (M[\pi^{-1}x/x])$ for any raw M, where $[\pi^{-1}x/x]$ indicates that x is replaced by $\pi^{-1}x$.*

Substituting N_1, ..., N_n for free occurrences of the distinct variables x_1, ..., x_n in the raw term M yields another raw term, which we denote by $M\{N_1, \ldots, N_n/x_1, \ldots, x_n\}$ or by $M\{\boldsymbol{N_i/x_i}\}$. The "usual" recursive definition for "ordinary" λ-terms (see, for example, [18]) carries over to NLC, and we omit the formal definition. However, for the base cases on suspensions we define

$$\begin{array}{lll} (\pi y)\{N_1, \ldots, N_n/x_1, \ldots, x_n\} =_{def} \pi y & \qquad & (\forall i)(x_i \neq y) \\ (\pi y)\{N_1, \ldots, N_n/x_1, \ldots, x_n\} =_{def} \pi * N_{i_0} & \qquad & (\exists i)(x_i = y) \text{ with } x_{i_0} = y \end{array}$$

Note the critical use of the object-level permutation action. Note also the crucial connection—used in many proofs—between suspension-substitutions and simultaneous substitution, which is easily proved by induction:

Lemma C.2 *For any term M we have $M[\pi^{-1}x/x] = M\{\pi^{-1}x/x\}$*

The next lemma expresses the meta-level action in terms of the object-level action. It is used to prove properties of both NLC and our categorical semantics.

Lemma C.3 ($\cdot$ in terms of $*$) *For any term M and $\{x_1, ..., x_n\} \subseteq Var$ with $fv(M) \subseteq \{x_1, ..., x_n\}$ we have $\pi \cdot M = (\pi * M)\{\pi^{-1}x_1/x_1, ..., \pi^{-1}x_n/x_n\}$.*

We use two definitions of α-equivalence. One is founded on capture avoiding substitution; the other on variable swapping. Each definition generates the same relation $\sim_\alpha \subset Term_{Sg} \times Term_{Sg}$ (see [11]).

The first definition [18] takes $\sim_\alpha$ to be the smallest equivalence relation closed under the congruence rules (for application and abstraction terms) and the axiom $\lambda^{\overline{a}}x : s.\, M \sim_\alpha \lambda^{\overline{a}}x' : s.\, M\{x'/x\}$ where $x' \notin var(M)$. The second definition is given in terms of variable swapping [11,14]. If $x, y \in Var$ then we define $(x\, y) \bullet M$ to be M in which any occurrence of x is swapped with y (and vice-versa). Then we can define $\sim_\alpha$ by the rules in Table C.1.

$$\frac{}{\pi x \sim_\alpha \pi x}\ (x \in Var \quad \pi \in Perm) \qquad\qquad \frac{M_1 \sim_\alpha M_1' \qquad M_2 \sim_\alpha M_2'}{M_1 M_2 \sim_\alpha M_1' M_2'}$$

$$\frac{(z\, x) \bullet M_1 \sim_\alpha (z\, y) \bullet M_2}{\lambda^{\overline{a}}x : s.\, M_1 \sim_\alpha \lambda^{\overline{a}}y : s.\, M_1}\ (z \notin var(M_1) \cup var(M_2))$$

Table C.1
Alpha Equivalence by Variable Swapping

It can easily be shown that $\sim_\alpha$ is equivariant for the permutation actions, that is $M \sim_\alpha N$ implies $\pi \cdot M \sim_\alpha \pi \cdot N$ and respectively for the object level permutation action. From this a well-defined permutation action on α-equivalence classes of terms is induced: $\pi \cdot [M]_\alpha \overset{\text{def}}{=} [\pi \cdot M]_\alpha$ and $\pi * [M]_\alpha \overset{\text{def}}{=} [\pi * M]_\alpha$

Lemma C.4 *Capture avoiding substitution lifts to the set of α-equivalence classes of terms, ${}^{Term_{Sg}}/_{\sim_\alpha}$, a nominal set under the meta-level permutation action on α-equivalence classes, with $supp([M]_\alpha) = supp(M)$.*

D Basic Properties of FM-cccs

We will use the functor $(-) \Rightarrow (+) : \mathcal{C}^{op} \times \mathcal{C} \to \mathcal{C}$, which is defined by $(A, B) \mapsto A \Rightarrow B$ and $(f, g) \mapsto f \Rightarrow g \overset{\text{def}}{=} \lambda(g \circ ev \circ (id_{A\Rightarrow B} \times f))$. An auxiliary lemma is used in establishing that our semantics is sound; the proof is routine category theory. Its use is illustrated briefly on page 111.

Lemma D.1

(i) *For any $f : A \times B \to C$ we have $\pi \cdot \lambda(f) = \lambda(\pi \cdot f)$*

(ii) $\pi_B \circ ev_{A,B} = ev_{\pi\cdot A,\pi\cdot B} \circ (\pi_{A\Rightarrow B} \times \pi_A)$.

(iii) $\pi \cdot (f \Rightarrow g) = \pi \cdot f \Rightarrow \pi \cdot g$

(iv) $\pi_{A\Rightarrow B} = \pi^{-1}_{\pi\cdot A} \Rightarrow \pi_B$

(v) *For any $f : A \times B \to C$ we have $\pi_{B\Rightarrow C} \circ \lambda(f) = \lambda(\pi_C \circ f \circ (id \times \pi^{-1}_{\pi\cdot B}))$*

(vi) *For any $f : A \times B \to C$ and $g : A' \to A$, $\lambda(f) \circ g = \lambda(f \circ (g \times id_B))$*

E Example Proofs

We include full details of a small but illustrative number of proofs. The idea is to give a flavour of the work that underpins our paper, and to provide evidence that the underlying proofs have been checked thoroughly.

Proof of Proposition 2.2:

Proof. It is easy to see that the meta-level (conjugation) mapping is a finitely supported permutation action, with specified support set. To show that the object-level (left multiplication) mapping is a permutation action, induct on the size of M, appealing to Lemma C.1 in the abstraction case. □

Proof of Lemma C.3:

Proof. Proof by induction on the structure of term M of

$$(\forall\pi)(\forall\{x_1, \ldots, x_n\})(fv(M) \subseteq \{x_1 \ldots x_n\}$$
$$\implies \pi \cdot M = (\pi * M)\{\pi^{-1}x_1/x_1, ..., \pi^{-1}x_n/x_n\}$$

We assume Lemma C.2 throughout.

SUSP: When M is τx_i the result follows immediately by the definition of substitution and the permutation actions.

CONST: Follows immediately.

LAM-ABS: Case M is $\lambda^{\overline{a}} x : s.\, M'$ where $fv(\lambda^{\overline{a}} x : s.\, M') \stackrel{\text{def}}{=} fv(M') \setminus \{x\} \subseteq \{x_1 \ldots x_n\}$. We examine the case when x is not an x_i; if x is an x_i the details are not too dissimilar. So for the induction step $fv(M') \subseteq \{x, x_1 \ldots, x_n\}$.

$$
\begin{aligned}
&\pi \cdot (\lambda^{\overline{a}} x : s.M') \\
&\quad \stackrel{\text{def}}{=} \lambda^{\pi \cdot \overline{a}} x : \pi \cdot s.\pi \cdot M' \\
&\quad = \lambda^{\pi \cdot \overline{a}} x : \pi \cdot s.(\pi * M')\{\pi^{-1}x/x, \boldsymbol{\pi^{-1} x_i / x_i}\} && \text{(induction)} \\
&\quad = \lambda^{\pi \cdot \overline{a}} x : \pi \cdot s.((\pi * M')\{\pi^{-1}x/x\})\{\boldsymbol{\pi^{-1} x_i / x_i}\} && (x \neq x_i) \\
&\quad = \lambda^{\pi \cdot \overline{a}} x : \pi \cdot s.(\pi * (M'\{\pi^{-1}x/x\}))\{\boldsymbol{\pi^{-1} x_i / x_i}\} && \text{(Lemma C.1)} \\
&\quad = (\lambda^{\pi \cdot \overline{a}} x : \pi \cdot s.\pi * (M'\{\pi^{-1}x/x\}))\{\boldsymbol{\pi^{-1} x_i / x_i}\} && (x \neq x_i \text{ so no capture}) \\
&\quad \stackrel{\text{def}}{=} (\pi * (\lambda^{\overline{a}} x : s.M'))\{\boldsymbol{\pi^{-1} x_i / x_i}\}
\end{aligned}
$$

APP: Case M is $N\ N'$.

$$
\begin{aligned}
&\pi \cdot (N\ N') \\
&\quad \stackrel{\text{def}}{=} (\pi \cdot N)\ (\pi \cdot N')) \\
&\quad = ((\pi * N)\{\boldsymbol{\pi^{-1} x_i / x_i}\})\ ((\pi * N')\{\boldsymbol{\pi^{-1} x_i / x_i}\}) && \text{(induction)} \\
&\quad \stackrel{\text{def}}{=} ((\pi * N)\ (\pi * N'))\{\boldsymbol{\pi^{-1} x_i / x_i}\} && \text{(def subst)} \\
&\quad = (\pi * (N\ N'))\{\boldsymbol{\pi^{-1} x_i / x_i}\}
\end{aligned}
$$

□

Proof of Lemma 3.3:

Proof. Since permutations are isomorphisms we only need to prove one direction of the implication. We have to prove, by (mutual) induction over the rules in Table A.1 and A.2,

$$
\begin{aligned}
&(\forall Th \rhd \nabla' \vdash^E [M]_\alpha : s') \quad [\\
&\qquad (\forall\ \nabla, \overline{a}, \pi, x, s) \quad (\nabla' \equiv \nabla, \overline{a} \mathrel{\#} x : s \\
&\qquad\qquad \Longrightarrow Th \rhd \nabla, \pi \cdot \overline{a} \mathrel{\#} x : \pi \cdot s \vdash^E [M\{\pi^{-1}x/x\}]_\alpha : s')) \quad]
\end{aligned}
$$

$$
\begin{aligned}
&(\forall Th \rhd \nabla' \vdash^E [M]_\alpha \approx [M']_\alpha : s') \quad [\\
&\qquad (\forall\ \nabla, \overline{a}, \pi, x, s) \quad (\nabla' \equiv \nabla, \overline{a} \mathrel{\#} x : s \\
&\qquad\qquad \Longrightarrow Th \rhd \nabla, \pi \cdot \overline{a} \mathrel{\#} x : \pi \cdot s \vdash^E [M\{\pi^{-1}x/x\}]_\alpha \approx [M'\{\pi^{-1}x/x\}]_\alpha : s')) \quad]
\end{aligned}
$$

In the remainder of this example proof we concentrate only on illustrating the care we take over dealing with proofs involving capture avoiding re-naming.

Rule (ABS) : To save any confusion over variable names consider the locally scoped instance of the rule

$$
\frac{\nabla', \overline{b} \mathrel{\#} y : t \vdash^E [N]_\alpha : t'}{\nabla' \vdash^E [\lambda^{\overline{b}} \boxed{y} : t.\, N]_\alpha : t^{\overline{b}} \Rightarrow t'}\ \textsf{ABS}
$$

in which the lambda bound $\boxed{y}$ (indicated by the box) is now in local scope. Consider the local instantiation of $(\forall\ \nabla, \overline{a}, \pi, x, s)$ when $x \overset{\text{def}}{=} y$ (and the other names remain the same). Thus we have $\nabla' \equiv \nabla, \overline{a} \# y : s$. For Induction Property Closure we have to prove that

$$\nabla, \pi \cdot \overline{a} \# y : \pi \cdot s \vdash^{E} [(\lambda^{\overline{b}} y : t.\, N)\{\pi^{-1} y / y\}]_{\alpha} = [\lambda^{\overline{b}} y : t.\, N]_{\alpha} : t' \quad (\diamond)$$

We cannot immediately invert ABS since the binding $\boxed{y}$ occurs in ∇'. Choosing distinct y' we have $[\lambda^{\overline{b}} y : t.\, N]_{\alpha} = [\lambda^{\overline{b}} y' : t.\, (y'\ y) \bullet N]_{\alpha}$ so we may now invert ABS to get

$$\nabla, \overline{a} \# y : s, \overline{b} \# y' : t \vdash^{E} [(y'\ y) \bullet N]_{\alpha} : t'$$

and hence by the variable equivariance of judgements, Lemma 3.2,

$$\nabla, \overline{a} \# y' : s, \overline{b} \# y : t \vdash^{E} [N]_{\alpha} : t'$$

Therefore by induction with $(\forall\ \nabla, \overline{a}, \pi, x, s)$ locally instantiated to $\nabla, \overline{b} \# y : t, \overline{a}, \pi, y', s$ we have

$$\nabla, \pi \cdot \overline{a} \# y' : \pi \cdot s, \overline{b} \# y : t \vdash^{E} [N\{\pi^{-1} y' / y'\}]_{\alpha} = [N]_{\alpha} : t'$$

since $y' \notin \mathit{var}(N)$. Hence by Lemma 3.2 we obtain $(\diamond)$ from

$$\nabla, \pi \cdot \overline{a} \# y : \pi \cdot s, \overline{b} \# y' : t \vdash^{E} [(y'\ y) \bullet N]_{\alpha} : t'$$

□

Proof of Proposition 3.4:

Proof. We prove by mutual induction over the rules in Figure A.1 and A.2 the following statements:

$$(\forall \mathit{Th} \rhd \nabla \vdash^{E} [M]_{\alpha} : s) \quad (\mathit{Th} \rhd \nabla \vdash^{E} [\pi * M]_{\alpha} : \pi \cdot s)$$

$$(\forall \mathit{Th} \rhd \nabla \vdash^{E} [M]_{\alpha} \approx [M']_{\alpha} : s) \quad (\mathit{Th} \rhd \nabla \vdash^{E} [\pi * M]_{\alpha} \approx [\pi * M']_{\alpha} : \pi \cdot s)$$

We give full details of Induction Property Closure for a few of the rules:

ABS: Suppose that $\nabla \vdash^{E} \lambda^{\overline{a}} x : s.\, M : s^{\overline{a}} \Rightarrow s'$. We consider the case where x is not present in the context. We can now directly deduce that $\nabla, \overline{a} \# x : s \vdash^{E} M : s'$ holds. By induction we obtain $\nabla, \overline{a} \# x : s \vdash^{E} \pi * M : \pi \cdot s'$. We then apply Lemma 3.3 to get $\nabla, \pi \cdot \overline{a} \# x : \pi \cdot s \vdash^{E} (\pi * M)\{\pi^{-1} x / x\} : \pi \cdot s$. Using the ABS-rule, we obtain $\nabla \vdash^{E} \lambda^{\pi \cdot \overline{a}} x : \pi \cdot s.\, (\pi * M)\{\pi^{-1} x / x\} : (\pi \cdot s)^{\pi \cdot \overline{a}} \Rightarrow \pi \cdot s'$. Then by applying Proposition 2.4 and Lemma C.2 we are done.

AP: Suppose that $\nabla \vdash^{E} F\ A : s'$. From this we can directly deduce $\nabla \vdash^{E} F : s^{\overline{a}} \Rightarrow s'$ and $\nabla \vdash^{E} \overline{a} \# A : s$. We now have to show that $\nabla \vdash^{E} \pi * (F\ A) : \pi \cdot s'$ holds, that is $\nabla \vdash^{E} (\pi * F)\ (\pi * A) : \pi \cdot s'$. By the AP-rule, this can be deduced by demonstrating that $\nabla \vdash^{E} \pi * F : \pi \cdot s^{\pi \cdot \overline{a}} \Rightarrow \pi \cdot s'$ and $\nabla \vdash^{E} \pi \cdot \overline{a} \# \pi * A : \pi \cdot s$ hold.

The former follows immediately by induction on $\nabla \vdash^{E} F : s^{\overline{a}} \Rightarrow s'$. For the latter, we have to demonstrate that

$$\nabla^{\#\overline{z}} \vdash^{E} (\pi \cdot \overline{a}\,\overline{z}) * (\pi * A) \approx \pi * A : \pi \cdot s$$

which by Lemma 3.3 and a couple of applications of Proposition 2.4 holds just in case

$$(\pi \cdot \nabla)^{\#\overline{z}} \vdash^{E} (\pi \cdot \overline{a}\,\overline{z}) * (\pi * (A\{\boldsymbol{\pi^{-1} x_i / x_i}\})) \approx \pi * (A\{\boldsymbol{\pi^{-1} x_i / x_i}\}) : \pi \cdot s \qquad (\diamond)$$

We show that $(\diamond)$ holds. Choose $\overline{z} \# (\pi \cdot \overline{a}, \pi \cdot \nabla, \pi * (A\{\boldsymbol{\pi^{-1} x_i / x_i}\}))$. From Lemma C.3 we have $\pi * (A\{\boldsymbol{\pi^{-1} x_i / x_i}\}) = \pi \cdot A$ and so $\pi^{-1} \cdot \overline{z} \# (\overline{a}, \nabla, A)$. With this, the definition of $\nabla \vdash^{E} \overline{a} \# A : s$ yields the equation

$$\nabla^{\#\pi^{-1}\cdot\overline{z}} \vdash^{E} (\overline{a}\,(\pi^{-1} \cdot \overline{z})) * A \approx A : s$$

and by induction $\nabla^{\#\pi^{-1}\cdot z} \vdash^{E} \pi * ((\overline{a}\,(\pi^{-1} \cdot z)) * A) \approx \pi * A : \pi \cdot s$. Note that for *Perm* we have the following general result: $\pi \circ (c\,d) = (\pi \cdot c\, \pi \cdot d) \circ \pi$. Using this result and Proposition 2.2 that $*$ is a permutation action, we can deduce that

$$\nabla^{\#\pi^{-1}\cdot\overline{z}} \vdash^{E} (\pi \cdot \overline{a}\,\overline{z}) * (\pi * A) \approx \pi * A : \pi \cdot s$$

holds. Applying Lemma 3.3, and then using Proposition 2.4 to associate brackets to the right, we obtain $(\diamond)$ and the argument is complete.

BF: Suppose that $\nabla \vdash^{E} (\lambda^{\overline{a}} x : s.\, M)\; N \approx M\{N/x\} : s'$. Considering the case where x is not present in the context, we have $\nabla, \overline{a} \# x : s \vdash^{E} M : s'$ and $\nabla \vdash^{E} \overline{a} \# N : s$. The typed expression $\nabla, \overline{a} \# x : s \vdash^{E} \pi * M : \pi \cdot s'$ is obtained by induction. We then apply Lemma 3.3 to obtain

$$\nabla, \pi \cdot \overline{a} \# x : \pi \cdot s \vdash^{E} (\pi * M)\{\pi^{-1} x/x\} : \pi \cdot s'.$$

Using the same reasoning as in the AP-case, we have $\nabla \vdash^{E} \pi \cdot \overline{a} \# \pi * N : \pi \cdot s$ by induction on $\nabla \vdash^{E} \overline{a} \# N : s$. We can now apply the BF-rule to obtain

$$\nabla \vdash^{E} (\lambda^{\pi\cdot\overline{a}} x : \pi \cdot s.\, (\pi * M)\{\pi^{-1} x/x\})\; (\pi * N) \approx ((\pi * M)\{\pi^{-1} x/x\})\{\pi * N/x\} : \pi \cdot s'$$

and since $M\{\pi^{-1} x/x\}\{\pi * N/x\} = M\{N/x\}$ we have

$$\nabla \vdash^{E} (\lambda^{\pi\cdot\overline{a}} x : \pi \cdot s.\, (\pi * M)\{\pi^{-1} x/x\})\; (\pi * N) \approx (\pi * M)\{N/x\} : \pi \cdot s'$$

Using Proposition 2.4 and the definition of the permutation action we are done.

EF: Suppose $x \notin fv(M)$ and $\nabla \vdash^{E} \lambda^{\overline{a}} x : s.\, (M\; x) \approx M : s^{\overline{a}} \Rightarrow s'$. From this we can directly deduce that $x \notin fv(\pi * M)$ and $\nabla \vdash^{E} M : s^{\overline{a}} \Rightarrow s'$. It then follows by induction that $\nabla \vdash^{E} \pi * M : (\pi \cdot s)^{\pi\cdot\overline{a}} \Rightarrow \pi \cdot s'$ holds. We conclude by applying the EF-rule along with a short calculation using the definition of the object-level permutation action.

The details for the remaining rules are similar. □

Proof of Theorem 4.2:

Proof. This proof does proceed by a mutual induction establishing the satisfaction of all judgement forms. Induction Property Closure for all the rules in Table A.1 and Table A.2 is similar to our example:

(AP): We need to show that $\llbracket \nabla \vdash^E F\ A : s' \rrbracket \Downarrow$ $(\diamond)$. By induction we have $\llbracket \nabla \vdash^E F : s^{\overline{a}} \Rightarrow s' \rrbracket \blacktriangleright f$ (1). Recalling that satisfaction of the freshness assertion is the satisfaction of an equation

$$\nabla \vdash^E \overline{a} \# A : s \quad \overset{\text{def}}{=} \quad (\forall / \exists \overline{z} \# (\nabla, \overline{a}, A)) \quad (\nabla^{\# \overline{z}} \vdash^E A \approx (\overline{a}\,\overline{z}) * A : s)$$

we have $\llbracket \nabla^{\# \overline{z}} \vdash^E A : s \rrbracket \blacktriangleright \theta$ and $\llbracket \nabla^{\# \overline{z}} \vdash^E (\overline{a}\,\overline{z}) * A : s \rrbracket \blacktriangleright \theta'$ with $\theta = \theta'$. Hence by Lemma 4.3 vi we have $\theta = \alpha \circ i$ where $\llbracket \nabla \vdash^E A : s \rrbracket \blacktriangleright \alpha$ and by Lemma 4.3 iii and 4.3 vi we have $\theta' = (\overline{a}\,\overline{z})_{\llbracket s \rrbracket} \circ \alpha \circ i$. From the (Epi When Fresh) property of FM-cccs we have $\alpha = (\overline{a}\,\overline{z})_{\llbracket s \rrbracket} \circ \alpha$, that is $\dagger(\alpha, \overline{a})$. Hence $\llbracket \nabla \vdash^E \overline{a} \# A : s \rrbracket \blacktriangleright \alpha^*$ (2). From (1) and (2) we have $(\diamond)$, with definition $ev \circ \langle f, \alpha^* \rangle$.

Property Closure for the rules in Table A.2 is trivial for (REF) (SYM) (TRANS). (WEAK) uses Lemma 4.3 v. (AE) uses Lemma 4.3 vi and Lemma 4.3 ii. (PERM) uses Lemma 4.3 iii and Lemma 4.3 vi. (BF) is quite similar to the details given for (AP).□

Proof of Lemma 4.4 part i and ii:

Proof.

(i) Following the definitions in our paper together with the properties of a perm-category, we have

$$\begin{aligned}
\pi \cdot \llbracket \nabla \rrbracket &= \pi \cdot (\llbracket s_1 \rrbracket^{\# \overline{a_1}} \times \ldots \times \llbracket s_n \rrbracket^{\# \overline{a_n}}) \\
&= (\pi \cdot \llbracket s_1 \rrbracket^{\# \overline{a_1}} \times \ldots \times \pi \cdot \llbracket s_n \rrbracket^{\# \overline{a_n}}) \\
&= ((\pi \cdot \llbracket s_1 \rrbracket)^{\# \pi \cdot \overline{a_1}} \times (\pi \cdot \llbracket s_n \rrbracket)^{\# \pi \cdot \overline{a_n}}) \\
&= (\llbracket \pi \cdot s_1 \rrbracket^{\# \pi \cdot \overline{a_1}} \times \llbracket \pi \cdot s_n \rrbracket^{\# \pi \cdot \overline{a_n}}) \\
&= \llbracket [\pi \cdot \overline{a_1} \# x_1 : \pi \cdot s_1, \ldots, \pi \cdot \overline{a_n} \# x_n : \pi \cdot s_n] \rrbracket \\
&= \llbracket \pi \cdot \nabla \rrbracket
\end{aligned}$$

(ii) Proof by induction on the structure of M

$$(\forall M) \quad [\quad (\forall\, \nabla, \pi, s) \quad (\pi \cdot \llbracket \nabla \vdash^E M : s \rrbracket \asymp \llbracket \pi \cdot \nabla \vdash^E \pi \cdot M : \pi \cdot s \rrbracket) \quad]$$

SUSP: It directly follows from the categorical semantics that

$$\llbracket \pi \cdot \nabla, \pi \cdot \overline{a} \# x : \pi \cdot s \vdash^E \pi \cdot \pi' x : \pi \cdot \pi' \cdot s \rrbracket \Downarrow$$

and $\llbracket \nabla, \overline{a} \# x : s \vdash^E \pi' x : \pi' \cdot s \rrbracket \Downarrow$

The equality follows by basic properties of FM-cccs.

CONST: It is immediate that $\llbracket \nabla \vdash^E c : s \rrbracket \Downarrow$ and $\llbracket \pi \cdot \nabla \vdash^E \pi \cdot c : \pi \cdot s \rrbracket \Downarrow$. The equality follows from the fact that $\llbracket - \rrbracket : Fun_\Sigma \to ob\, \mathcal{C}$ is equivariant.

LAM-ABS: Suppose $\llbracket \pi \cdot \nabla \vdash^E \pi \cdot (\lambda^{\overline{a}} x : s.\, M) : \pi \cdot (s^{\overline{a}} \Rightarrow s') \rrbracket \Downarrow$ and it is equal to f_π. By the definition of the meta-level permutation action and the inductively defined semantics we get

$$[\![\pi \cdot \nabla \vdash^E \lambda^{\pi \cdot \overline{a}} x : \pi \cdot s .\, \pi \cdot M : (\pi \cdot s)^{\pi \cdot \overline{a}} \Rightarrow \pi \cdot s']\!] \blacktriangleright \lambda(m_\pi)$$

for some m_π where $[\![\pi \cdot \nabla, \pi \cdot \overline{a} \# x : \pi \cdot s \vdash^E \pi \cdot M : \pi \cdot s']\!] \blacktriangleright m_\pi$. By induction we deduce that $[\![\nabla, \overline{a} \# x : s \vdash^E M : s']\!] \blacktriangleright m$ such that $\pi \cdot m = m_\pi$. We then apply the rule for semantics of abstraction to obtain $[\![\nabla \vdash^E \lambda^{\overline{a}} x : s .\, M : s^{\overline{a}} \Rightarrow s']\!] \blacktriangleright \lambda(m)$, that is, $[\![\nabla \vdash^E \lambda^{\overline{a}} x : s .\, M : s^{\overline{a}} \Rightarrow s']\!]\Downarrow$. The definitional existence proof in the converse direction follows by similar reasoning. We now need to show that $f_\pi = \pi \cdot \lambda(m)$.

$$\begin{aligned} f_\pi &\stackrel{\text{def}}{=} \lambda(m_\pi) & \\ &= \lambda(\pi \cdot m) && \text{(induction)} \\ &= \pi \cdot \lambda(m) && \text{(Lemma D.1 (i))} \end{aligned}$$

APP: Suppose $[\![\pi \cdot \nabla \vdash^E \pi \cdot (F\ A) : \pi \cdot s']\!]\Downarrow$ and it is equal to t_π. By the definition of the meta-level permutation action and the rule for semantics of applications

$$[\![\pi \cdot \nabla \vdash^E (\pi \cdot F)\ (\pi \cdot A) : \pi \cdot s']\!] \blacktriangleright ev \circ \langle f_\pi, \theta_\pi \rangle$$

for

$$[\![\pi \cdot \nabla \vdash^E \pi \cdot F : (\pi \cdot s)^{\pi \cdot \overline{a}} \Rightarrow \pi \cdot s']\!] \blacktriangleright f_\pi$$

and

$$[\![\pi \cdot \nabla \vdash^E \pi \cdot \overline{a} \# \pi \cdot A : \pi \cdot s]\!] \blacktriangleright \theta_\pi .$$

We have $[\![\pi \cdot \nabla \vdash^E \pi \cdot \overline{a} \# \pi \cdot A : \pi \cdot s]\!] \blacktriangleright {\alpha_\pi}^*$ by the rule for freshness assertion semantics where $[\![\pi \cdot \nabla \vdash^E \pi \cdot A : \pi \cdot s]\!] \blacktriangleright \alpha_\pi$ such that $\dagger(\pi \cdot \overline{a}, \alpha_\pi)$. Given that $\blacktriangleright$ is a partial function, we have that $\theta_\pi = {\alpha_\pi}^*$. By induction we get $[\![\nabla \vdash^E F : s^{\overline{a}} \Rightarrow s']\!] \blacktriangleright f$ and $[\![\nabla \vdash^E A : s]\!] \blacktriangleright \alpha$ such that $f_\pi = \pi \cdot f$ and $\alpha_\pi = \pi \cdot \alpha$. We now deduce from $\dagger(\pi \cdot \overline{a}, \alpha_\pi)$ that $\dagger(\overline{a}, \alpha)$ holds: Let $\overline{z} \# (\overline{a}, \alpha)$. It follows immediately that $\pi \cdot \overline{z} \# (\pi \cdot \overline{a}, \pi \cdot \alpha)$ and hence from $\dagger(\pi \cdot \overline{a}, \pi \cdot \alpha)$ we obtain equation (E.1). In the equations below, we write internal permutation actions τ_C as $\tau_\_$ since the source-target data does not play a significant role in our reasoning, and indeed is probably obfuscating:

$$(\pi \cdot \overline{z}\, \pi \cdot \overline{a})_\_ \circ (\pi \cdot \alpha) \circ i = (\pi \cdot \alpha) \circ i \tag{E.1}$$

$$(\pi \cdot \overline{z}\, \pi \cdot \overline{a})_\_ \circ \pi_\_ \circ \alpha \circ \pi_\_^{-1} \circ i = \pi_\_ \circ \alpha \circ \pi_\_^{-1} \circ i \tag{E.2}$$

$$\pi_\_ \circ (\overline{z}\, \overline{a})_\_ \circ \alpha \circ \pi_\_^{-1} \circ i = \pi_\_ \circ \alpha \circ \pi_\_^{-1} \circ i \tag{E.3}$$

$$(\overline{z}\, \overline{a})_\_ \circ \alpha \circ i \circ \pi_\_^{-1} = \alpha \circ i \circ \pi_\_^{-1} \tag{E.4}$$

$$(\overline{z}\, \overline{a})_\_ \circ \alpha \circ i = \alpha \circ i \tag{E.5}$$

By definition of the FM-ccc permutation action on morphisms we obtain equation (E.2). The transposition notation $(\overline{z}\, \overline{a})$ is short for $(z_1\, a_1) \circ \ldots \circ (z_k\, a_k)$.

Since in $Perm$, $\pi \circ (c\, d) = (\pi(c)\, \pi(d)) \circ \pi$ holds generally for single transpositions $(c\, d)$, and since permutation actions satisfy $(\tau' \circ \tau)_C = \tau'_{\tau \cdot C} \circ \pi_C$ we have

$$\pi\_ \circ (\overline{z}\, \overline{a})\_ = (\pi \circ (\overline{z}\, \overline{a}))\_ = ((\pi \cdot \overline{z}\, \pi \cdot \overline{a}) \circ \pi)\_ = (\pi \cdot \overline{z}\, \pi \cdot \overline{a})\_ \circ \pi\_$$

This gives us equation (E.3). Any internal permutation action $(\tau_C : C \to \tau \cdot C \mid C \in ob\, \mathcal{C})$ is a natural transformation $Id \to \tau \cdot -$ and in particular so is π_{-}^{-1}. Since also π_{-} is iso, equation (E.4) holds. Finally since π_{-}^{-1} is iso we obtain (E.5). Hence, $\dagger(\overline{a}, \alpha)$ holds.

We can now apply the rule for freshness assertion semantics to obtain $\llbracket \nabla \vdash^E \overline{a} \# A : s \rrbracket \blacktriangleright \alpha^*$, followed by the rule for application semantics to get $\llbracket \nabla \vdash^E F\ A : s' \rrbracket \blacktriangleright ev \circ \langle f, \alpha^* \rangle$. Hence, we have $\llbracket \nabla \vdash^E F\ A : s' \rrbracket \Downarrow$. The definitional existence proof in the converse direction follows by similar reasoning.

We now show that $t_\pi = \pi \cdot (ev \circ \langle f, \alpha^* \rangle)$. Note that $(\pi \cdot \alpha)^* = \pi \cdot \alpha^* \quad (\Phi)$ holds: This follows immediately from the universal property of inclusion image restriction and the definition of $\pi \cdot -$. Hence

$$\begin{aligned}
t_\pi &\overset{\text{def}}{=} ev_{(\llbracket \pi \cdot s \rrbracket^{\# \pi \cdot \overline{a}}, \llbracket \pi \cdot s' \rrbracket)} \circ \langle f_\pi, \theta_\pi \rangle && \\
&= ev_{(\llbracket \pi \cdot s \rrbracket^{\# \pi \cdot \overline{a}}, \llbracket \pi \cdot s' \rrbracket)} \circ \langle f_\pi, {\alpha_\pi}^* \rangle && (\theta_\pi = {\alpha_\pi}^*) \\
&= ev_{(\pi \cdot (\llbracket s \rrbracket^{\# \overline{a}}), \pi \cdot \llbracket s' \rrbracket)} \circ \langle \pi \cdot f, (\pi \cdot \alpha)^* \rangle && (\text{induction}) \\
&= ev_{(\pi \cdot (\llbracket s \rrbracket^{\# \overline{a}}), \pi \cdot \llbracket s' \rrbracket)} \circ \langle \pi \cdot f, \pi \cdot \alpha^* \rangle && (\Phi) \\
&= ev_{(\pi \cdot (\llbracket s \rrbracket^{\# \overline{a}}), \pi \cdot \llbracket s' \rrbracket)} \circ (\pi \cdot \langle f, \alpha^* \rangle) && (\text{equivariant products}) \\
&= (\pi \cdot ev_{(\llbracket s \rrbracket^{\# \overline{a}}, \llbracket s' \rrbracket)}) \circ (\pi \cdot \langle f, \alpha^* \rangle) && (\text{equivariant exponentials}) \\
&= \pi \cdot (ev_{(\llbracket s \rrbracket^{\# \overline{a}}, \llbracket s' \rrbracket)} \circ \langle f, \alpha^* \rangle) && (\text{equivariance of } \circ)
\end{aligned}$$

□

Proof of Proposition 5.1:

Proof. Objects: $ob\, Cl(Th)$ is the set of freshness environments. In our construction, which slightly modifies Clouston's [7], we will make use of the following typical objects

$$\nabla \overset{\text{def}}{=} (\overline{a}_1 \# x_1 : s_1, ..., \overline{a}_n \# x_n : s_n) \quad \nabla' \overset{\text{def}}{=} (\overline{a'}_1 \# x'_1 : s'_1, ..., \overline{a'}_m \# x'_m : s'_m)$$
$$\text{and } \nabla'' \overset{\text{def}}{=} (\overline{a''}_1 \# x''_1 : s''_1, ..., \overline{a''}_k \# x''_k : s''_k)$$

Morphisms: Morphisms $\delta \overset{\text{def}}{=} ([M_1]_\approx, \ldots, [M_m]_\approx) : \nabla \to \nabla'$ are lists of typed expressions such that for $1 \leq i \leq m$ we have $Th \rhd \nabla \vdash^E \overline{a}'_i \# M_i : s'_i$, and $[M_i]_\approx$ is the equivalence class of those T such that $Th \rhd \nabla \vdash^E M_i \approx T : s'_i$.

Identity: For ∇ as above, the identity morphism is given by $([x_1]_\approx, \ldots, [x_n]_\approx)$.

Composition: Let $\delta : \nabla \to \nabla'$ be as above and let $\delta' \overset{\text{def}}{=} ([N_1]_\approx \ldots [N_k]_\approx) : \nabla' \to \nabla''$. Then

$$\delta' \circ \delta \overset{\text{def}}{=} ([N_1\{\boldsymbol{M_i}/\boldsymbol{x_i}\}]_\approx, \ldots, [N_k\{\boldsymbol{M_i}/\boldsymbol{x_i}\}]_\approx)$$

FM-Category Structure: Please see [7].

Equivariant Exponentials: For objects ∇ and ∇', the exponential is defined by

$$\nabla \Rightarrow \nabla' \stackrel{\text{def}}{=} (\overline{a}'_1 \mathrel{\#} f_1 : \boldsymbol{(\text{И}a'_1.\, s_i)^{a_i}} \Rightarrow s'_1, ..., \overline{a}'_m \mathrel{\#} f_m : \boldsymbol{(\text{И}a'_m.\, s_i)^{a_i}} \Rightarrow s'_m)$$

where for each $j = 1, ..., m$, $\boldsymbol{(\text{И}a'_j.\, s_i)^{a_i}} \Rightarrow s'_j \stackrel{\text{def}}{=} (\text{И}a'_j.\, s_1)^{\overline{a}_1} \Rightarrow ((\text{И}a'_j.\, s_2)^{\overline{a}_2} \Rightarrow ...((\text{И}a'_j.\, s_n)^{\overline{a}_n} \Rightarrow s'_j))$. The evaluation map $ev : (\nabla \Rightarrow \nabla') \times \nabla \to \nabla'$ is $([f_1 \;\; \boldsymbol{\xi_i}]_{\approx}, \ldots, [f_m \;\; \boldsymbol{\xi_i}]_{\approx})$ where $f_j \;\; \boldsymbol{\xi_i}$ is shorthand for the NLC-application $(...((f_j \;\; (\langle a' \rangle\, (a'\, a'_j)x_1)) \;\; (\langle a' \rangle\, (a'\, a'_j)x_i))...) \;\; (\langle a' \rangle\, (a'\, a'_j)x_n)$. Finally, given $\delta \stackrel{\text{def}}{=} ([M_1]_{\approx}, \ldots, [M_m]_{\approx}) : \nabla'' \times \nabla \to \nabla'$ the exponential mate $\lambda(\delta) : \nabla'' \to (\nabla \Rightarrow \nabla')$ is given by

$$\begin{aligned}(&\lambda^{\overline{a}_1} y_1 : \text{И}a'_1.\, s_1. ...\lambda^{\overline{a}_n} y_n : \text{И}a'_1.\, s_n. M_1\{y_1 \mathbin{@} a'_1 ... y_n \mathbin{@} a'_1 / x_1 ... x_n\}, \ldots, \\ &\qquad \lambda^{\overline{a}_1} y_1 : \text{И}a'_m.\, s_1. ...\lambda^{\overline{a}_n} y_n : \text{И}a'_m.\, s_1. M_m\{y_1 \mathbin{@} a'_m ... y_n \mathbin{@} a'_m / x_1 ... x_n\})\end{aligned}$$

The verification that $Cl(Th)$ is indeed a ИFM-ccc is omitted. □

Proof of Proposition 5.3:

Proof. We need to check that both structure maps (of Sg) are equivariant. This is easy: for example $\pi \cdot \llbracket c \rrbracket_{\mathcal{G}} \stackrel{\text{def}}{=} \pi \cdot ([c]_{\approx}) \stackrel{\text{def}}{=} ([\pi \cdot c]_{\approx}) \stackrel{\text{def}}{=} \llbracket \pi \cdot c \rrbracket_{\mathcal{G}}$

The rest of the proof is by induction on the structure of M.

SUSP: We have

$$\llbracket \nabla \vdash \pi x_i : \pi \cdot s_i \rrbracket_{\mathcal{G}} \blacktriangleright \pi_{\llbracket s_i \rrbracket_{\mathcal{G}}} \circ i^{\overline{a}_i}_{\llbracket s_i \rrbracket_{\mathcal{G}}} \circ pr_i : \nabla \to \llbracket s_i \rrbracket_{\mathcal{G}}^{\#\overline{a}_i} \to \llbracket s_i \rrbracket_{\mathcal{G}} \to \pi \cdot \llbracket s_i \rrbracket_{\mathcal{G}}$$

Expanding the definitions we get

$$\nabla \xrightarrow{([x_i]_{\approx})} (\overline{a}_i \mathrel{\#} x_i : s_i) \xrightarrow{([x_i]_{\approx})} (\emptyset \mathrel{\#} x_i : s_i) \xrightarrow{([\pi x_i]_{\approx})} (\emptyset \mathrel{\#} v : \pi \cdot s_i)$$

with the composition being $([\pi x_i]_{\approx})$ as required.

APP: Suppose that $Th \rhd \nabla \vdash^E F\ A : s' \quad (\diamond\diamond)$. We wish to prove that $\llbracket \nabla \vdash^E F\ A : s' \rrbracket_{\mathcal{G}} \blacktriangleright ([F\ A]_{\approx})$ on the inductive assumptions that

$$\llbracket \nabla \vdash^E F : s^{\overline{a}} \Rightarrow s' \rrbracket_{\mathcal{G}} \blacktriangleright ([F]_{\approx}) : \llbracket \nabla \rrbracket_{\mathcal{G}} \to (\emptyset \mathrel{\#} v : s^{\overline{a}} \Rightarrow s')$$
$$\llbracket \nabla \vdash^E A : s \rrbracket_{\mathcal{G}} \blacktriangleright ([A]_{\approx}) : \llbracket \nabla \rrbracket_{\mathcal{G}} \to (\emptyset \mathrel{\#} v : s)$$

We claim that $\llbracket \nabla \vdash^E \overline{a} \mathrel{\#} A : s \rrbracket_{\mathcal{G}} \blacktriangleright ([A]_{\approx})^* \quad (\diamond)$. To check $\dagger(\overline{a}, ([A]_{\approx}))$ consider, for completely distinct $\overline{z}$, the morphism[3]

$$\nabla^{\#\overline{z}} \xrightarrow{([x_i]_{\approx})} \nabla \xrightarrow{([A]_{\approx})} (\emptyset \mathrel{\#} v : s) \xrightarrow{([(\overline{a}\,\overline{z})v]_{\approx})} (\emptyset \mathrel{\#} v : s)$$

Now $(\diamond)$ holds provided that

$$((([(\overline{a}\,\overline{z})v]_{\approx}) \circ ([A]_{\approx})) \circ ([x_i]_{\approx}) \stackrel{\text{def}}{=} ([((\overline{a}\,\overline{z})v)\{A/v\}]_{\approx}) \circ ([x_i]_{\approx}) = ([A]_{\approx}) \circ ([x_i]_{\approx})$$

[3] In fact $(\overline{a}\,\overline{z}) \cdot s = s$ for technical reasons that we omit for lack of space, so the morphism does have the given target object.

that is $([(\overline{a}\,\overline{z}) * A]_{\approx}) = ([A]_{\approx})$. This states that $\nabla^{\#\overline{z}} \vdash^{E} (\overline{a}\,\overline{z}) * A \approx A : s$ or equivalently $\nabla \vdash^{E} \overline{a} \# A : s$ which itself follows from rule AP inversion of $(\diamond\diamond)$. □

Proof of Proposition 5.4:

Proof. It is easy to see that the meta-level (conjugation) mapping is a finitely supported permutation action, with specified support set. To show that the object-level (left multiplication) mapping is a permutation action, induct on the size of M, appealing to Lemma C.1 in the abstraction case. □

Proof of Theorem 5.5:

Proof. If $\nabla \vdash^{E} M \approx M' : s$ is satisfied by all models, then it is also satisfied by the generic model $\mathcal{G}$ of Th in $Cl(Th)$. From Proposition 5.4 we have $([M]_{\approx}) = ([M']_{\approx})$. Hence, $Th \rhd \nabla \vdash^{E} M \approx M' : s$. □

Continuity of Gödel's System T Definable Functionals via Effectful Forcing

Martín Escardó[1]

School of Computer Science
University of Birmingham
Birmingham, England

Abstract

It is well-known that the Gödel's system T definable functions $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ are continuous, and that their restrictions from the Baire type $(\mathbb{N} \to \mathbb{N})$ to the Cantor type $(\mathbb{N} \to 2)$ are uniformly continuous. We offer a new, relatively short and self-contained proof. The main technical idea is a concrete notion of generic element that doesn't rely on forcing, Kripke semantics or sheaves, which seems to be related to generic effects in programming. The proof uses standard techniques from programming language semantics, such as dialogues, monads, and logical relations. We write this proof in intensional Martin-Löf type theory (MLTT) from scratch, in Agda notation. Because MLTT has a computational interpretation and Agda can be seen as a programming language, we can run our proof to compute moduli of (uniform) continuity of T-definable functions.

Keywords: Gödel's system T, continuity, uniform continuity, Baire space, Cantor space, intensional Martin-Löf theory, Agda, dialogue, semantics, logical relation.

1 Introduction

This is a relatively short, and self-contained, proof of the well-known fact that any function $f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ that is definable in Gödel's system T is continuous, and that its restriction from the Baire type $(\mathbb{N} \to \mathbb{N})$ to the Cantor type $(\mathbb{N} \to 2)$ is uniformly continuous [15,2]. We believe the proof is new, although it is related to previous work discussed below. The main technical idea is a concrete notion of generic element that doesn't rely on forcing, Kripke semantics or sheaves, which seems to be related to generic effects in programming [13]. Several well-known ideas from logic, computation, constructive mathematics and programming-language semantics naturally appear here, in a relatively simple, self-contained, and hopefully appealing, development.

The idea is to represent a function $f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ by a well-founded dialogue tree, and extract continuity information about f from this tree. To calculate such a tree from a system T term t: $(\iota \Rightarrow \iota) \Rightarrow \iota$ denoting f, we work with an auxiliary

[1] Email: m.escardo@cs.bham.ac.uk

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.com/locate/entcs

interpretation of system T, which gives a function $\tilde{f} : (\tilde{\mathbb{N}} \to \tilde{\mathbb{N}}) \to \tilde{\mathbb{N}}$, where $\tilde{\mathbb{N}}$ is the set of dialogue trees. Applying $\tilde{f}$ to a certain *generic sequence* $\tilde{\mathbb{N}} \to \tilde{\mathbb{N}}$, the desired dialogue tree is obtained. We now explain this idea in more detail.

In the set-theoretical model of system T, the ground type ι is interpreted as the set $\mathbb{N}$ of natural numbers, and if the types σ and τ are interpreted as sets X and Y, then the type $\sigma \Rightarrow \tau$ is interpreted as the set of all functions $X \to Y$. We consider an auxiliary model that replaces the interpretation of the ground type by the set $\tilde{\mathbb{N}}$, but keeps the interpretation of $\Rightarrow$ as the formation of the set of all functions. In this model, the zero constant is interpreted by a suitable element $\tilde{0}$ of $\tilde{\mathbb{N}}$, the successor constant is interpreted by a function $\tilde{\mathbb{N}} \to \tilde{\mathbb{N}}$, and each iteration combinator is interpreted by a function $(X \to X) \to X \to \tilde{\mathbb{N}} \to X$. An element of the set $\tilde{\mathbb{N}}$ is a well-founded dialogue tree that describes the computation of a natural number relative to an unspecified oracle $\alpha : \mathbb{N} \to \mathbb{N}$. An internal node is labeled by a natural number representing a query to the oracle, and has countably many branches corresponding to the possible answers. Each leaf is labeled by a natural number and represents a possible outcome of the computation. These dialogues represent computations in the sense of Kleene [10].

If a particular oracle $\alpha : \mathbb{N} \to \mathbb{N}$ is given, we get a natural number from any $d \in \tilde{\mathbb{N}}$ via a decodification function

$$\text{decode} : (\mathbb{N} \to \mathbb{N}) \to \tilde{\mathbb{N}} \to \mathbb{N}.$$

It turns out that there is a function

$$\text{generic} : \tilde{\mathbb{N}} \to \tilde{\mathbb{N}}$$

that can be regarded as a *generic sequence* in the sense that, for any particular sequence $\alpha : \mathbb{N} \to \mathbb{N}$,

$$\begin{tikzcd}
\tilde{\mathbb{N}} \arrow[r, "\text{generic}"] \arrow[d, "\text{decode } \alpha"'] & \tilde{\mathbb{N}} \arrow[d, "\text{decode } \alpha"] \\
\mathbb{N} \arrow[r, "\alpha"'] & \mathbb{N}.
\end{tikzcd}$$

That is, the generic sequence codes any concrete sequence α, provided the sequence α itself is used as the concrete oracle for decodification. The idea is that the application of the function generic to a dialogue tree adds a new layer of choices at its leaves.

Next we show that for any given term $t : (\iota \Rightarrow \iota) \Rightarrow \iota$ denoting a function $f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ in the standard interpretation and $\tilde{f} : (\tilde{\mathbb{N}} \to \tilde{\mathbb{N}}) \to \tilde{\mathbb{N}}$ in the dialogue interpretation, we have that

$$f\ \alpha = \text{decode}\ \alpha\ (\tilde{f}\ \text{generic}).$$

This is proved by establishing a logical relation between the set-theoretic and dialogue models. Thus we can compute a dialogue tree of f by applying $\tilde{f}$ to the generic sequence.

The set $\tilde{\mathbb{N}}$ is constructed as $\mathrm{B}\,\mathbb{N}$ for a suitable dialogue monad B. Then the interpretation of the constant zero is $\eta\ 0$ where η is the unit of the monad, the interpretation of the successor constant is given by functoriality as B succ, and the interpretation of the primitive recursion constant is given by the Kleisli extension of its standard interpretation. The object part $\mathrm{B}\ X$ of the monad is inductively defined by the constructors

$$\eta : X \to \mathrm{B}\ X,$$
$$\beta : (\mathbb{N} \to \mathrm{B}\ X) \to \mathbb{N} \to \mathrm{B}\ X,$$

where η constructs leaves and β constructs a tree $\beta\ \phi\ n$ given countably many trees ϕ and a label n. With $X = \mathbb{N}$, we have $\beta\ \eta : \mathbb{N} \to \mathrm{B}\ \mathbb{N}$, and the generic sequence is the Kleisli extension of $\beta\ \eta$. Thus, the generic sequence seems to be a sort of *generic effect* in the sense of [13]. Notice that our interpretation is a call-by-name version of Moggi's semantics.

Using this, it follows that if a function $f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ is the set-theoretical interpretation of some system T term $t : (\iota \Rightarrow \iota) \Rightarrow \iota$, then it is continuous and its restriction to $\mathbb{N} \to 2$ is uniformly continuous, where 2 is the set with elements 0 and 1. The reason is that a dialogue produces an answer after finitely many queries, because it is well-founded, and that a dialogue tree for a function $(\mathbb{N} \to 2) \to \mathbb{N}$ is finite, because it is finitely branching. Recall that continuity means that, for any sequence of integers $\alpha : \mathbb{N} \to \mathbb{N}$, there is $m : \mathbb{N}$, called a modulus of continuity of f at the point α, such that any sequence α' that agrees with α at the first m positions gives the same result, that is, $f\ \alpha = f\ \alpha'$. Uniform continuity means that there is $m : \mathbb{N}$, called a modulus of uniform continuity of f on $\mathbb{N} \to 2$, such that any two binary sequences α and α' that agree at the first m positions give the same result.

Our arguments are constructive, and we write the full proof from scratch in intensional Martin-Löf type theory (MLTT), in Agda notation [4], without the use of libraries. We don't assume previous familiarity with Agda, but we do require rudimentary knowledge of MLTT. The Agda source file for this program/proof [7] is written in Knuth's *literate* style, which automatically generates the LaTeX file that produces this article. Agda both checks proofs and can run them. Notice that MLTT or Agda cannot prove or disprove that all functions $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ are continuous, as they are compatible with both classical and constructive mathematics, like Bishop mathematics [3]. The theorem here is that certain functions $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ are continuous: those that can be defined in system T.

Related work. The idea of computing continuity information by applying a function to effectful arguments goes back to Longley [11], who passes exceptions to the function. A similar approach is described in an example given by Bauer and Pretnar [1].

The idea of working with computation trees is of course very old, going back to Brouwer [2] in intuitionistic mathematics, and to Kleene [10] in computability theory in the form of dialogues, where the input is referred to as an oracle. Howard [9] derives computation trees for system T, by operational methods, by successively reducing a term so that each time an oracle given by a free variable of type $\iota \Rightarrow \iota$ is queried, countably many branches of the computation are created, corresponding to the possible answers given by the oracle. Hancock and Setzer use variations of dialogue trees to describe interactive computation in type theory [12] (see also [8]).

Our work is directly inspired by Coquand and Jaber's work on forcing in type theory [5,6]. Like Howard, they derive computation trees by operational methods.

They extend dependent type theory with a constant for a generic element, and then decorate judgements with subscripts that keep track of approximation information about the generic element as the computations proceed (similarly to [15]). In this way they extract continuity information. They prove the termination and soundness of this modification of type theory using Tait's computability method, which here is manifested as a logical relation between our two models. They also provide a Haskell implementation for the system T case as an appendix, which uses a monad that is the composition of the list monad (for nondeterminism) and of the state monad. Their Haskell program implements a normalization procedure with bookkeeping information, tracked by the monad, that produces computation trees. Because they only account for uniform continuity in their Haskell implementation, such trees are finite. They describe their work as a computational interpretation of forcing and continuity as presented in Beeson [2]. The difference is that their approach is syntactical whereas ours is semantical, and the reader may sense an analogy with normalization by evaluation. Notice that these arguments only show that the *definable* functions are continuous. To get a constructive model in which *all* functions are continuous, they work with iterated forcing, which is related to our recent work [16], but this is another story.

Organization. (2) Formal proof in Agda. (3) Informal, rigorous proof.

Acknowledgements. I benefitted from remarks on a previous version of this paper by Thierry Coquand, Dan Ghica, Achim Jung, Chuangjie Xu, and the anonymous referees.

2 Proof in Martin-Löf type theory in Agda notation

2.1 Agda preliminaries

The purpose of this subsection is two-fold: (1) To develop a tiny Agda library for use in the following sections, and (2) to briefly explain Agda notation [4] for MLTT. We assume rudimentary knowledge of (intensional) Martin-Löf type theory and the BHK interpretation of the quantifiers as products Π and sums Σ. We don't use any feature of Agda that goes beyond standard MLTT. If we were trying to be purist, we would use W-types rather than some of our inductive definitions using the Agda keyword *data*. Notice that the coloured text in the electronic version of this paper is the Agda code.

The universe of all types is denoted by Set, and types are called sets (this is a universe à la Russell). Products Π are denoted by $\forall$ in Agda. Consider the definition of the (interpretation of) the standard combinators:

```
Ķ : ∀{X Y : Set} → X → Y → X
Ķ x y = x
Ş : ∀{X Y Z : Set} → (X → Y → Z) → (X → Y) → X → Z
Ş f g x = f x (g x)
```

The curly braces around the set variables indicate that these are implicit parameters, to be inferred by Agda whenever Ķ and Ş are used. If Agda fails to uniquely infer the missing arguments, one has to write e.g. Ķ $\{X\}$ $\{Y\}$ x y rather than the abbreviated form Ķ x y. The following should be self-explanatory:

```
_∘_ : ∀{X Y Z : Set} → (Y → Z) → (X → Y) → (X → Z)
g ∘ f = λ x → g(f x)

data ℕ : Set where
  zero : ℕ
  succ : ℕ → ℕ
rec : ∀{X : Set} → (X → X) → X → ℕ → X
rec f x  zero    = x
rec f x (succ n) = f(rec f x n)
```

Agda has a termination checker that verifies that recursions are well-founded, and hence all functions are total. We also need types of binary digits, finite lists, and finite binary trees:

```
data ℕ₂ : Set where
  ₀ ₁ : ℕ₂

data List (X : Set) : Set where
  []   : List X
  _::_ : X → List X → List X

data Tree (X : Set) : Set where
  empty : Tree X
  branch : X → (ℕ₂ → Tree X) → Tree X
```

Sums are not built-in and hence need to be defined:

```
data Σ {X : Set} (Y : X → Set) : Set where
  _,_ : ∀(x : X)(y : Y x) → Σ {X} Y
```

The definition says that an element of $\Sigma\ \{X\}\ Y$ is a pair (x,y) with $x : X$ and $y : Y\ x$. Notice that comma is not a reserved symbol: we define it as a binary operator to construct dependent pairs. Because $Y = \lambda(x : X) \to Y\ x$ if one assumes the η-law, and because the first argument is implicit, we can write $\Sigma\ \{X\}\ Y$ as $\Sigma\ \mathrm{Y}$ or $\Sigma \setminus(x : X) \to Y\ x$, where backslash is the same thing as lambda. We will use backslash exclusively for sums.

```
π₀ : ∀{X : Set} {Y : X → Set} → (Σ \(x : X) → Y x) → X
π₀(x , y) = x
π₁ : ∀{X : Set} {Y : X → Set} → ∀(t : Σ \(x : X) → Y x) → Y(π₀ t)
π₁(x , y) = y
```

The identity type Id $X\ x\ y$ is written $x \equiv y$ with X implicit, and is inductively defined as “the least reflexive relation”:

```
data _≡_ {X : Set} : X → X → Set where
  refl : ∀{x : X} → x ≡ x

sym : ∀{X : Set} → ∀{x y : X} → x ≡ y → y ≡ x
sym refl = refl
trans : ∀{X : Set} → ∀{x y z : X} → x ≡ y → y ≡ z → x ≡ z
trans refl refl = refl
cong : ∀{X Y : Set} → ∀(f : X → Y) → ∀{x₀ x₁ : X} → x₀ ≡ x₁ → f x₀ ≡ f x₁
cong f refl = refl
cong₂ : ∀{X Y Z : Set} → ∀(f : X → Y → Z)
      → ∀{x₀ x₁ : X}{y₀ y₁ : Y} → x₀ ≡ x₁ → y₀ ≡ y₁ → f x₀ y₀ ≡ f x₁ y₁
cong₂ f refl refl = refl
```

2.2 Dialogues and continuity

We consider the computation of functionals $(X \to Y) \to Z$ with dialogue trees. We work with the following inductively defined type of (well founded) dialogue trees indexed by three types X, Y and Z. These are Y-branching trees with X-labeled internal nodes and Z-labeled leaves:

```
data D (X Y Z : Set) : Set where
   η : Z → D X Y Z
   β : (Y → D X Y Z) → X → D X Y Z
```

A leaf is written $\eta\ z$, and it gives the final answer z (η will be the unit of a monad). A forest is a Y-indexed family ϕ of trees. Given such a forest ϕ and $x : X$, we can build a new tree β ϕ x whose root is labeled by x, which has a subtree ϕ y for each $y : Y$. We can imagine $x : X$ as query, for which an oracle α gives some intermediate answer $y = \alpha\ x$: Y. After this answer y, we move to the subtree ϕ y, and the dialogue proceeds in this way, until a leaf with the final answer is reached:

```
dialogue : ∀{X Y Z : Set} → D X Y Z → (X → Y) → Z
dialogue (η z)     α = z
dialogue (β ϕ x) α = dialogue (ϕ(α x)) α
```

We say that a function $(X \to Y) \to Z$ is *eloquent* if it is computed by some dialogue:

```
eloquent : ∀{X Y Z : Set} → ((X → Y) → Z) → Set
eloquent f = Σ \d → ∀ α → dialogue d α ≡ f α
```

Here we are interested in the case $X{=}Y{=}Z{=}\mathbb{N}$. Think of functions $\alpha : \mathbb{N} \to \mathbb{N}$ as sequences of natural numbers. The set of such sequences is called the Baire space:

```
Baire : Set
Baire = ℕ → ℕ
```

Functions Baire → ℕ are coded by a particular kind of dialogue trees, namely B ℕ where B is defined as follows:

```
B : Set → Set
B = D ℕ ℕ
```

We work with a refined version of continuity, which gives more information than the traditional notion introduced in Section 1, where the modulus of continuity is a finite list of indices rather than an upper bound for the indices. The agreement relation determined by a list of indices is inductively defined as follows, where $\alpha \equiv[\ s\ ]\ \alpha'$ says that the sequences α and α' agree at the indices collected in the list s:

(i) $\alpha \equiv[\ []\ ]\ \alpha'$,

(ii) $\alpha\ i \equiv \alpha'\ i \to \alpha \equiv[\ s\ ]\ \alpha' \to \alpha \equiv[\ i :: s\ ]\ \alpha'$.

We write this inductive definition as follows in Agda, where we give the name [] to the proof of the first clause and the name :: to the proof of the second clause, that is, using the same constructor names as for the inductively defined type of lists:

```
data _≡[_]_ {X : Set} : (ℕ → X) → List ℕ → (ℕ → X) → Set where
   []    : ∀{α α’ : ℕ → X} → α ≡[ [] ] α’
   _::_ : ∀{α α’ : ℕ → X}{i : ℕ}{s : List ℕ} → α i ≡ α’ i → α ≡[ s ] α’ → α ≡[ i :: s ] α’

continuous : (Baire → ℕ) → Set
continuous f = ∀(α : Baire) → Σ \(s : List ℕ) → ∀(α’ : Baire) → α ≡[ s ] α’ → f α ≡ f α’
```

It is an easy exercise, left to the reader, to produce an Agda proof that this refined notion of continuity implies the traditional notion of continuity, by taking the maximum value of the list s. Functions defined by dialogues are continuous, because a dialogue produces an answer after finitely many queries:

```
dialogue-continuity : ∀(d : B ℕ) → continuous(dialogue d)
dialogue-continuity (η n) α = ([] , lemma)
   where
      lemma : ∀ α' → α ≡[ [] ] α' → n ≡ n
      lemma α' r = refl
dialogue-continuity (β ϕ i) α = ((i :: s) , lemma)
   where
      IH : ∀(i : ℕ) → continuous(dialogue(ϕ(α i)))
      IH i = dialogue-continuity (ϕ(α i))
      s : List ℕ
      s = π₀(IH i α)
      claim₀ : ∀(α' : Baire) → α ≡[ s ] α' → dialogue(ϕ(α i)) α ≡ dialogue(ϕ(α i)) α'
      claim₀ = π₁(IH i α)
      claim₁ : ∀(α' : Baire) → α i ≡ α' i → dialogue (ϕ (α i)) α' ≡ dialogue (ϕ (α' i)) α'
      claim₁ α' r = cong (λ n → dialogue (ϕ n) α') r
      lemma : ∀(α' : Baire) → α ≡[ i :: s ] α'   → dialogue (ϕ(α i)) α ≡ dialogue(ϕ (α' i)) α'
      lemma α' (r :: rs) = trans (claim₀ α' rs) (claim₁ α' r)
```

This formal proof is informally explained as follows. We show that

$$\forall(d : \mathrm{B}\ \mathbb{N}) \to \mathrm{continuous}(\mathrm{dialogue}\ d)$$

by induction on d. Expanding the definition, this amounts to, using Agda notation,

$$\forall\ d \to \forall\ \alpha \to \Sigma\ \backslash s \to \forall\ \alpha' \to \alpha \equiv\![\ s\ ]\ \alpha' \to \mathrm{dialogue}\ d\ \alpha \equiv \mathrm{dialogue}\ d\ \alpha'.$$

For the base case $d = \eta\ n$, the definition of the function dialogue gives dialogue $d\ \alpha = n$, and so we must show that, for any α,

$$\Sigma\ \backslash s \to \forall\ \alpha' \to \alpha \equiv\![\ s\ ]\ \alpha' \to n \equiv n.$$

We can take $s = []$ and then we are done, because $n \equiv n$ by reflexivity. This is what the first equation of the formal proof says. Thus notice that Agda, in accordance with MLTT, silently expands definitions by reduction to normal form. For the induction step $d = \beta\ \phi\ i$, expanding the definition of the dialogue function, what we need to prove is that, for an arbitrary α,

$$\Sigma\ \backslash s' \to \forall\ \alpha' \to \alpha \equiv\![\ s'\ ]\ \alpha' \to \mathrm{dialogue}\ (\phi(\alpha\ i))\ \alpha \equiv \mathrm{dialogue}\ (\phi\ \alpha'\ i)\ \alpha'.$$

The induction hypothesis is $\forall(i : \mathbb{N}) \to \mathrm{continuous}(\mathrm{dialogue}(\phi(\alpha\ i)))$, which gives, for any i and our arbitrary α,

$$\Sigma\ \backslash s \to \forall\ \alpha' \to \alpha \equiv\![\ s\ ]\ \alpha' \to \mathrm{dialogue}(\phi(\alpha\ i))\ \alpha = \mathrm{dialogue}(\phi(\alpha\ i))\ \alpha'.$$

Using the two projections π_0 and π_1 we get s and a proof that

$$\forall\ \alpha' \to \alpha \equiv\![\ s\ ]\ \alpha' \to \mathrm{dialogue}(\phi(\alpha\ i))\ \alpha = \mathrm{dialogue}(\phi(\alpha\ i))\ \alpha'.$$

Hence we can take $s' = i :: s$, and the desired conclusion holds substituting equals for equals (with cong) using transitivity and the definition $\alpha\ i \equiv \alpha'\ i \to \alpha \equiv\![\ s\ ]\ \alpha' \to \alpha \equiv\![\ i :: s\ ]\ \alpha'$. This amounts to the second equation of the proof, where in the pattern of the proof of the lemma we have $r : \alpha\ i \equiv \alpha'\ i$ and $rs : \alpha \equiv\![\ s\ ]\ \alpha'$.

We need the following technical lemma because it is not provable in intensional MLTT that any two functions are equal if they are pointwise equal. The proof is admitedly written in a rather laconic form. The point is that the notion of continuity depends only on the values of the function, and the hypothesis says that the two functions have the same values. Notice that the axiom of function extensionality (any two pointwise equal functions are equal) is not false but rather not provable or disprovable, and is consistent [14].

```
continuity-extensional : ∀(f g : Baire → ℕ) → (∀ α → f α ≡ g α) → continuous f → continuous g
continuity-extensional f g t c α = (π₀(c α) , (λ α' r → trans (sym (t α)) (trans (π₁(c α) α' r) (t α'))))
eloquent-is-continuous : ∀(f : Baire → ℕ) → eloquent f → continuous f
eloquent-is-continuous f (d , e) = continuity-extensional (dialogue d) f e (dialogue-continuity d)
```

The development for uniform continuity is similar to the above, with the crucial difference that a dialogue tree in C ℕ is finite:

```
Cantor : Set
Cantor = ℕ → ℕ₂
C : Set → Set
C = D ℕ ℕ₂
```

We work with a refined version of uniform continuity (cf. Section 1), where the modulus is a finite binary tree s of indices rather than an upper bound of the indices. We could have worked with a list of indices, but the proofs are shorter and more direct using trees. The agreement relation defined by a tree of indices is inductively defined as follows, where $\alpha \equiv[[\ s\ ]]\ \alpha'$ says that α and α' agree at the positions collected in the tree s:

```
data _≡[[_]]_ {X : Set} : (ℕ → X) → Tree ℕ → (ℕ → X) → Set where
  empty : ∀{α α' : ℕ → X} → α ≡[[ empty ]] α'
  branch :
    ∀{α α' : ℕ → X}{i : ℕ}{s : ℕ₂ → Tree ℕ}
    → α i ≡ α' i → (∀(j : ℕ₂) → α ≡[[ s j ]] α') → α ≡[[ branch i s ]] α'
```

Again we are using the same constructor names as for the type of trees.

```
uniformly-continuous : (Cantor → ℕ) → Set
uniformly-continuous f = Σ \(s : Tree ℕ) → ∀(α α' : Cantor) → α ≡[[ s ]] α' → f α ≡ f α'

dialogue-UC : ∀(d : C ℕ) → uniformly-continuous(dialogue d)
dialogue-UC (η n) = (empty , λ α α' n → refl)
dialogue-UC (β φ i) = (branch i s , lemma)
  where
  IH : ∀(j : ℕ₂) → uniformly-continuous(dialogue(φ j))
  IH j = dialogue-UC (φ j)
  s : ℕ₂ → Tree ℕ
  s j = π₀(IH j)
  claim : ∀ j α α' → α ≡[[ s j ]] α' → dialogue (φ j) α ≡ dialogue (φ j) α'
  claim j = π₁(IH j)
  lemma : ∀ α α' → α ≡[[ branch i s ]] α' → dialogue (φ (α i)) α ≡ dialogue (φ (α' i)) α'
  lemma α α' (branch r l) = trans fact₀ fact₁
    where
    fact₀ : dialogue (φ (α i)) α ≡ dialogue (φ (α' i)) α
    fact₀ = cong (λ j → dialogue(φ j) α) r
    fact₁ : dialogue (φ (α' i)) α ≡ dialogue (φ (α' i)) α'
    fact₁ = claim (α' i) α α' (l(α' i))

UC-extensional : ∀(f g : Cantor → ℕ) → (∀(α : Cantor) → f α ≡ g α)
               → uniformly-continuous f → uniformly-continuous g
UC-extensional f g t (u , c) = (u , (λ α α' r → trans (sym (t α)) (trans (c α α' r) (t α'))))

eloquent-is-UC : ∀(f : Cantor → ℕ) → eloquent f → uniformly-continuous f
eloquent-is-UC f (d , e) = UC-extensional (dialogue d) f e (dialogue-UC d)
```

We finish this section by showing that the restriction of an eloquent function f : Baire → ℕ to the Cantor type is also eloquent. We first define a pruning function from B ℕ to C ℕ that implements restriction:

```
embed-ℕ₂-ℕ : ℕ₂ → ℕ
```

```
embed-ℕ₂-ℕ ₀ = zero
embed-ℕ₂-ℕ ₁ = succ zero

embed-C-B : Cantor → Baire
embed-C-B α = embed-ℕ₂-ℕ ∘ α

C-restriction : (Baire → ℕ) → (Cantor → ℕ)
C-restriction f = f ∘ embed-C-B

prune : B ℕ → C ℕ
prune (η n) = η n
prune (β φ i) = β (λ j → prune(φ(embed-ℕ₂-ℕ j))) i

prune-behaviour : ∀(d : B ℕ)(α : Cantor) → dialogue (prune d) α ≡ C-restriction(dialogue d) α
prune-behaviour (η n)     α = refl
prune-behaviour (β φ n) α = prune-behaviour (φ(embed-ℕ₂-ℕ(α n))) α

eloquent-restriction : ∀(f : Baire → ℕ) → eloquent f → eloquent(C-restriction f)
eloquent-restriction f (d , c) = (prune d , λ α → trans (prune-behaviour d α) (c (embed-C-B α)))
```

2.3 Gödel's system T extended with an oracle

For simplicity, we work with system T in its original combinatory form. This is no loss of generality, because both the combinatory and the lambda-calculus forms define the same elements of the set-theoretical model, and here we are interested in the continuity of the definable functionals. The system T type expressions and terms are inductively defined as follows:

```
data type : Set where
   ι    : type
   _⇒_ : type → type → type

data T : (σ : type) → Set where
   Zero    : T ι
   Succ    : T(ι ⇒ ι)
   Rec     : ∀{σ : type}       → T((σ ⇒ σ) ⇒ σ ⇒ ι ⇒ σ)
   K       : ∀{σ τ : type}     → T(σ ⇒ τ ⇒ σ)
   S       : ∀{ρ σ τ : type} → T((ρ ⇒ σ ⇒ τ) ⇒ (ρ ⇒ σ) ⇒ ρ ⇒ τ)
   _·_     : ∀{σ τ : type}     → T(σ ⇒ τ) → T σ → T τ

infixr 1 _⇒_
infixl 1 _·_
```

Notice that there are five constants (or combinators) and one binary constructor (application). Notice also that one can build only well-typed terms. The set-theoretical interpretation of type expressions and terms is given by

```
Set⟦_⟧ : type → Set
Set⟦ ι ⟧ = ℕ
Set⟦ σ ⇒ τ ⟧ = Set⟦ σ ⟧ → Set⟦ τ ⟧


⟦_⟧ : ∀{σ : type} → T σ → Set⟦ σ ⟧
⟦ Zero ⟧     = zero
⟦ Succ ⟧     = succ
⟦ Rec ⟧      = rec
⟦ K ⟧         = Ķ
⟦ S ⟧         = Ș
⟦ t · u ⟧    = ⟦ t ⟧ ⟦ u ⟧
```

An element of the set-theoretical model is called T-definable if there is a T-term denoting it:

```
T-definable : ∀{σ : type} → Set⟦ σ ⟧ → Set
```

```
T-definable x = Σ \t → ⟦ t ⟧ ≡ x
```

As discussed above, the main theorem, proved in the last subsection, is that every T-definable function Baire → ℕ is continuous. The system T type of such functionals is (ι ⇒ ι) ⇒ ι.

We also consider system T extended with a formal oracle Ω : ι ⇒ ι:

```
data TΩ : (σ : type) → Set where
    Ω      : TΩ(ι ⇒ ι)
    Zero   : TΩ ι
    Succ   : TΩ(ι ⇒ ι)
    Rec    : ∀{σ : type}     → TΩ((σ ⇒ σ) ⇒ σ ⇒ ι ⇒ σ)
    K      : ∀{σ τ : type}   → TΩ(σ ⇒ τ ⇒ σ)
    S      : ∀{ρ σ τ : type} → TΩ((ρ ⇒ σ ⇒ τ) ⇒ (ρ ⇒ σ) ⇒ ρ ⇒ τ)
    _·_    : ∀{σ τ : type}   → TΩ(σ ⇒ τ) → TΩ σ → TΩ τ
```

In the standard set-theoretical interpretation, the oracle can be thought of as a free variable ranging over elements of the interpretation Baire of the type expression ι ⇒ ι:

```
⟦_⟧' : ∀{σ : type} → TΩ σ → Baire → Set⟦ σ ⟧
⟦ Ω ⟧'        α = α
⟦ Zero ⟧'     α = zero
⟦ Succ ⟧'     α = succ
⟦ Rec ⟧'      α = rec
⟦ K ⟧'        α = Ķ
⟦ S ⟧'        α = Ş
⟦ t · u ⟧'    α = ⟦ t ⟧' α (⟦ u ⟧' α)
```

To regard TΩ as an extension of T we need to work with an embedding:

```
embed : ∀{σ : type} → T σ → TΩ σ
embed Zero = Zero
embed Succ = Succ
embed Rec = Rec
embed K = K
embed S = S
embed (t · u) = (embed t) · (embed u)
```

2.4 The dialogue interpretation of system T

We now consider an auxiliary interpretation of system T extended with an oracle in order to show that the original T-definable functions Baire → ℕ are continuous. In the alternative semantics, types are interpreted as the underlying objects of certain algebras of the dialogue monad. The ground type is interpreted as the free algebra of the standard interpretation, and function types as function sets. For the sake of brevity, we will include only the parts of the definition of the monad that we actually need for our purposes.

```
kleisli-extension : ∀{X Y : Set} → (X → B Y) → B X → B Y
kleisli-extension f (η x)     = f x
kleisli-extension f (β φ i) = β (λ j → kleisli-extension f (φ j)) i

B-functor : ∀{X Y : Set} → (X → Y) → B X → B Y
B-functor f = kleisli-extension(η ∘ f)
```

The following two lemmas are crucial. We first swap the two arguments of the dialogue function to have the view that from an element of the Baire type we get a B-algebra B X → X for any X:

```
decode : ∀{X : Set} → Baire → B X → X
decode α d = dialogue d α
```

The decodification map is natural for any oracle α : Baire:

$$\begin{tikzcd}
\mathrm{B}\ X \arrow[r, "\mathrm{B}\ g"] \arrow[d, "\text{decode } \alpha"'] & \mathrm{B}\ Y \arrow[d, "\text{decode } \alpha"] \\
X \arrow[r, "g"'] & Y.
\end{tikzcd}$$

```
decode-α-is-natural : ∀{X Y : Set}(g : X → Y)(d : B X)(α : Baire) → g(decode α d) ≡ decode α (B-functor g d)
decode-α-is-natural g (η x)     α = refl
decode-α-is-natural g (β φ i) α = decode-α-is-natural g (φ(α i)) α
```

The following diagram commutes for any $f : X \to \mathrm{B}\ Y$:

$$\begin{tikzcd}
\mathrm{B}\ X \arrow[rr, "\text{kleisli-extension } f"] \arrow[d, "\text{decode } \alpha"'] & & \mathrm{B}\ Y \arrow[d, "\text{decode } \alpha"] \\
X \arrow[r, "f"'] & \mathrm{B}\ Y \arrow[r, "\text{decode } \alpha"'] & Y.
\end{tikzcd}$$

```
decode-kleisli-extension : ∀{X Y : Set}(f : X → B Y)(d : B X)(α : Baire)
   → decode α (f(decode α d)) ≡ decode α (kleisli-extension f d)
decode-kleisli-extension f (η x)     α = refl
decode-kleisli-extension f (β φ i) α = decode-kleisli-extension f (φ(α i)) α
```

System TΩ type expressions are interpreted as the underlying sets of certain algebras of the dialogue monad. The base type is interpreted as the underlying set of the free algebra of the standard interpretation, and function types are interpreted as sets of functions, exploiting the fact that algebras are exponential ideals (if Y is the underlying object of an algebra, then so is the set of all functions $X \to Y$ for any X, with the pointwise structure).

```
B-Set⟦_⟧ : type → Set
B-Set⟦ ι ⟧ = B(Set⟦ ι ⟧)
B-Set⟦ σ ⇒ τ ⟧ = B-Set⟦ σ ⟧ → B-Set⟦ τ ⟧
```

According to the official definition of an algebra of a monad, to show that a set X is the underlying object of an algebra one must provide a structure map $\mathrm{B}\ X \to X$. Alternatively, which is more convenient for us, one can provide a generalized Kleisli extension operator, defined as follows, where the base case is just Kleisli extension, and the induction step is pointwise extension:

```
Kleisli-extension : ∀{X : Set} {σ : type} → (X → B-Set⟦ σ ⟧) → B X → B-Set⟦ σ ⟧
Kleisli-extension {X} {ι} = kleisli-extension
Kleisli-extension {X} {σ ⇒ τ} = λ g d s → Kleisli-extension {X} {τ} (λ x → g x s) d
```

With this we can now define the dialogue interpretation of system TΩ. The generic element of the Baire type under this interpretation will interpret the Baire oracle Ω:

```
generic : B ℕ → B ℕ
generic = kleisli-extension(β η)
```

As discussed in Section 1, the crucial property of the generic element is this:

$$\begin{tikzcd}
\mathrm{B}\ \mathbb{N} \arrow[r, "\text{generic}"] \arrow[d, "\text{decode } \alpha"'] & \mathrm{B}\ \mathbb{N} \arrow[d, "\text{decode } \alpha"] \\
\mathbb{N} \arrow[r, "\alpha"'] & \mathbb{N}.
\end{tikzcd}$$

```
generic-diagram : ∀(α : Baire)(d : B ℕ) → α(decode α d) ≡ decode α (generic d)
generic-diagram α (η n) = refl
generic-diagram α (β ϕ n) = generic-diagram α (ϕ(α n))
```

The alternative interpretations of zero and successor are obvious:

```
zero' : B ℕ
zero' = η zero
succ' : B ℕ → B ℕ
succ' = B-functor succ
```

And the interpretation of the primitive recursion combinator again uses Kleisli extension in an obvious way:

```
rec' : ∀{σ : type} → (B-Set⟦ σ ⟧ → B-Set⟦ σ ⟧) → B-Set⟦ σ ⟧ → B ℕ → B-Set⟦ σ ⟧
rec' f x = Kleisli-extension(rec f x)
```

This gives the dialogue interpretation. Notice that the interpretations of K, S and application are standard. This is because we interpret function types as sets of functions:

```
B⟦_⟧ : ∀{σ : type} → TΩ σ → B-Set⟦ σ ⟧
B⟦ Ω ⟧       = generic
B⟦ Zero ⟧    = zero'
B⟦ Succ ⟧    = succ'
B⟦ Rec ⟧     = rec'
B⟦ K ⟧       = Ḳ
B⟦ S ⟧       = Ṣ
B⟦ t · u ⟧   = B⟦ t ⟧ (B⟦ u ⟧)
```

This semantics gives the desired dialogue trees:

```
dialogue-tree : T((ι ⇒ ι) ⇒ ι) → B ℕ
dialogue-tree t = B⟦ (embed t) · Ω ⟧
```

The remainder of the development is the formulation and proof of the correctness of the dialogue-tree function. We conclude this section with the trivial proof that the embedding of T into TΩ preserves the standard interpretation and furthermore is independent of oracles:

```
preservation : ∀{σ : type} → ∀(t : T σ) → ∀(α : Baire) → ⟦ t ⟧ ≡ ⟦ embed t ⟧' α
preservation Zero α = refl
preservation Succ α = refl
preservation Rec α = refl
preservation K α = refl
preservation S α = refl
preservation (t · u) α = cong₂ (λ f x → f x) (preservation t α) (preservation u α)
```

2.5 Relating the two models

The main lemma is that for any term t : TΩ ι,

$$[\![t]\!]' \; \alpha \equiv \text{decode } \alpha \; (\text{B}[\![t]\!]).$$

We use the following logical relation to prove this:

```
R : ∀{σ : type} → (Baire → Set⟦ σ ⟧) → B-Set⟦ σ ⟧ → Set

R {ι} n n' =
  ∀(α : Baire) → n α ≡ decode α n'

R {σ ⇒ τ} f f' =
  ∀(x : Baire → Set⟦ σ ⟧)(x' : B-Set⟦ σ ⟧) → R {σ} x x' → R {τ} (λ α → f α (x α)) (f' x')
```

We need a (fairly general) technical lemma, which is used for constants with an interpretation using the Kleisli-extension operator. In our case, this is just the recursion combinator. The proof is by induction on type expressions, crucially relying on the lemma *decode-kleisli-extension*, but is routine otherwise:

```
R-kleisli-lemma : ∀(σ : type)(g : ℕ → Baire → Set⟦ σ ⟧)(g' : ℕ → B-Set⟦ σ ⟧)
  → (∀(k : ℕ) → R (g k) (g' k))
  → ∀(n : Baire → ℕ)(n' : B ℕ) → R n n' → R (λ α → g (n α) α) (Kleisli-extension g' n')

R-kleisli-lemma ι g g' rg n n' rn = λ α → trans (fact₃ α) (fact₀ α)
  where
    fact₀ : ∀ α → decode α (g' (decode α n')) ≡ decode α (kleisli-extension g' n')
    fact₀ = decode-kleisli-extension g' n'
    fact₁ : ∀ α → g (n α) α ≡ decode α (g'(n α))
    fact₁ α = rg (n α) α
    fact₂ : ∀ α → decode α (g' (n α)) ≡ decode α (g' (decode α n'))
    fact₂ α = cong (λ k → decode α (g' k)) (rn α)
    fact₃ : ∀ α → g (n α) α ≡ decode α (g' (decode α n'))
    fact₃ α = trans (fact₁ α) (fact₂ α)

R-kleisli-lemma (σ ⇒ τ) g g' rg n n' rn
  = λ y y' ry → R-kleisli-lemma τ (λ k α → g k α (y α)) (λ k → g' k y') (λ k → rg k y y' ry) n n' rn
```

The proof of the main lemma is by induction on terms, crucially relying on the lemmas *generic-diagram* (for the term Ω), *decode-is-natural* (for the term Succ) and *R-kleisli-lemma* (for the term Rec). The terms K and S are routine (but laborious and difficult to get right in an informal calculation), and so is the induction step for application:

```
main-lemma : ∀{σ : type}(t : TΩ σ) → R ⟦ t ⟧' (B⟦ t ⟧)

main-lemma Ω = lemma
  where
    claim : ∀ α n n' → n α ≡ dialogue n' α → α(n α) ≡ α(decode α n')
    claim α n n' s = cong α s
    lemma : ∀(n : Baire → ℕ)(n' : B ℕ) → (∀ α → n α ≡ decode α n')
      → ∀ α → α(n α) ≡ decode α (generic n')
    lemma n n' rn α = trans (claim α n n' (rn α)) (generic-diagram α n')

main-lemma Zero = λ α → refl
main-lemma Succ = lemma
  where
    claim : ∀ α n n' → n α ≡ dialogue n' α → succ(n α) ≡ succ(decode α n')
    claim α n n' s = cong succ s
    lemma : ∀(n : Baire → ℕ)(n' : B ℕ) → (∀ α → n α ≡ decode α n')
      → ∀ α → succ (n α) ≡ decode α (B-functor succ n')
    lemma n n' rn α = trans (claim α n n' (rn α)) (decode-α-is-natural succ n' α)


main-lemma {(σ ⇒ .σ) ⇒ .σ ⇒ ι ⇒ .σ} Rec = lemma
```

```
    where
      lemma : ∀(f : Baire → Set⟦ σ ⟧ → Set⟦ σ ⟧)(f' : B-Set⟦ σ ⟧ → B-Set⟦ σ ⟧) → R {σ ⇒ σ} f f'
          → ∀(x : Baire → Set⟦ σ ⟧)(x' : B-Set⟦ σ ⟧)
          → R {σ} x x' → ∀(n : Baire → ℕ)(n' : B ℕ) → R {ι} n n'
          → R {σ} (λ α → rec (f α) (x α) (n α)) (Kleisli-extension(rec f' x') n')
      lemma f f' rf x x' rx = R-kleisli-lemma σ g g' rg
          where
            g : ℕ → Baire → Set⟦ σ ⟧
            g k α = rec (f α) (x α) k
            g' : ℕ → B-Set⟦ σ ⟧
            g' k = rec f' x' k
            rg : ∀(k : ℕ) → R (g k) (g' k)
            rg zero = rx
            rg (succ k) = rf (g k) (g' k) (rg k)

main-lemma K = λ x x' rx y y' ry → rx

main-lemma S = λ f f' rf g g' rg x x' rx → rf x x' rx (λ α → g α (x α)) (g' x') (rg x x' rx)

main-lemma (t · u) = main-lemma t ⟦ u ⟧' B⟦ u ⟧ (main-lemma u)
```

This gives the correctness of the dialogue-tree function defined above: the standard interpretation of a term is computed by its dialogue tree.

```
dialogue-tree-correct : ∀(t : T((ι ⇒ ι) ⇒ ι))(α : Baire) → ⟦ t ⟧ α ≡ decode α (dialogue-tree t)
dialogue-tree-correct t α = trans claim₀ claim₁
    where
    claim₀ : ⟦ t ⟧ α ≡ ⟦ (embed t) · Ω ⟧' α
    claim₀ = cong (λ g → g α) (preservation t α)
    claim₁ : ⟦ (embed t) · Ω ⟧' α ≡ decode α (dialogue-tree t)
    claim₁ = main-lemma ((embed t) · Ω) α
```

The desired result follows directly from this:

```
eloquence-theorem : ∀(f : Baire → ℕ) → T-definable f → eloquent f
eloquence-theorem f (t , r) = (dialogue-tree t , λ α → trans(sym(dialogue-tree-correct t α))(cong(λ g → g α) r))

corollary₀ : ∀(f : Baire → ℕ) → T-definable f → continuous f
corollary₀ f d = eloquent-is-continuous f (eloquence-theorem f d)

corollary₁ : ∀(f : Baire → ℕ) → T-definable f → uniformly-continuous(C-restriction f)
corollary₁ f d = eloquent-is-UC (C-restriction f) (eloquent-restriction f (eloquence-theorem f d))
```

This concludes the full, self-contained, MLTT proof in Agda notation, given from scratch. Because MLTT proofs are programs, we can run the two corollaries to compute moduli of (uniform) continuity of T-definable functions. Because MLTT itself has an interpretation in ZF(C), in which types are sets in the sense of classical mathematics, the results of this paper hold in classical mathematics too. Because the LaTeX source for this article [7] is simultaneously an Agda file that type-checks, the readers don't need to check the routine details of the proofs themselves, provided they trust the minimal core of Agda used here, and can instead concentrate on the interesting details of the constructions and proofs. One can envisage a future in which it will be easier to write (constructive and non-constructive) formal proofs than informal, rigorous proofs, letting our minds concentrate on the insights. This is certainly a provocative statement. But, in fact, the proof presented here was directly written in its formal form, without an informal draft other than a mental picture starting from the idea of generic sequence as described in the introduction, with some rudimentary help by Agda to perform the routine steps. Tactic-based systems such as e.g. Coq provide much more help, which in some instances can be considered as non-routine even if ultimately they are based on algorithms. But our principal motivation for writing this formal proof in an MLTT or realizability based computer system such as NuPrl, Coq, Lego, Agda, Minlog etc. is that mentioned above, that the proof is literally a program too, and hence can be used to compute moduli of (uniform) continuity, without the need to write a separate algorithm based on an informal, rigorous proof, as it is usually currently done, including by ourselves in previous work.

Having said that, it is useful to have a self-contained informal rigorous proof, which we include in the next section. Before that, we conclude this section by running our formal constructive proof for the purposes of illustration.

2.6 Experiments

To illustrate the concrete sense in which the above formal proof is constructive, we develop some experiments. These experiments are not meant to indicate the usefulness of the theorem proved above. They merely make clear that the theorems do have a concrete computational content.

First of all, given a term $t : (\iota \Rightarrow \iota) \Rightarrow \iota$, we can compute its modulus of (uniform) continuity.

```
mod-cont : T((ι ⇒ ι) ⇒ ι) → Baire → List ℕ
mod-cont t α = π₀(corollary₀ ⟦ t ⟧ (t , refl) α)
mod-cont-obs : ∀(t : T((ι ⇒ ι) ⇒ ι))(α : Baire) → mod-cont t α ≡ π₀(dialogue-continuity (dialogue-tree t) α)
mod-cont-obs t α = refl

infixl 0 _::_
infixl 1 _++_
_++_ : {X : Set} → List X → List X → List X
[] ++ u = u
(x :: t) ++ u = x :: t ++ u
flatten : {X : Set} → Tree X → List X
flatten empty = []
flatten (branch x t) = x :: flatten(t 0) ++ flatten(t 1)

mod-unif : T((ι ⇒ ι) ⇒ ι) → List ℕ
mod-unif t = flatten(π₀ (corollary₁ ⟦ t ⟧ (t , refl)))
```

The following Agda declaration allows us to write e.g. 3 rather than succ(succ(succ zero)):

```
{-# BUILTIN NATURAL ℕ #-}
{-# BUILTIN ZERO zero #-}
{-# BUILTIN SUC succ #-}
```

A difficulty we face is that it is not easy to write system T programs in the combinatory version of system T. Hence we start by developing some machinery.

```
I : ∀{σ : type} → T(σ ⇒ σ)
I {σ} = S · K · (K {σ} {σ})
I-behaviour : ∀{σ : type}{x : Set⟦ σ ⟧} → ⟦ I ⟧ x ≡ x
I-behaviour = refl

number : ℕ → T ι
number zero = Zero
number (succ n) = Succ · (number n)
```

Here is our first example:

```
t₀ : T((ι ⇒ ι) ⇒ ι)
t₀ = K · (number 17)
```

```
t₀-interpretation : ⟦ t₀ ⟧ ≡ λ α → 17
t₀-interpretation = refl
example₀ example₀' : List ℕ
example₀ = mod-cont t₀ (λ i → i)
example₀' = mod-unif t₀
```

These examples both evaluate to []. To provide more sophisticated examples, we work with an impoverished context ɣ that allows us to consider just one free variable v, which is represented by the I combinator:

```
v : ∀{ɣ : type} → T(ɣ ⇒ ɣ)
v = I
```

Application for such a context amounts to the S combinator:

```
infixl 1 _•_
_•_ : ∀{ɣ σ τ : type} → T(ɣ ⇒ σ ⇒ τ) → T(ɣ ⇒ σ) → T(ɣ ⇒ τ)
f • x = S · f · x

Number : ∀{ɣ} → ℕ → T(ɣ ⇒ ι)
Number n = K · (number n)
```

Here is an example:

```
t₁ : T((ι ⇒ ι) ⇒ ι)
t₁ = v • (Number 17)
t₁-interpretation : ⟦ t₁ ⟧ ≡ λ α → α 17
t₁-interpretation = refl
example₁ : List ℕ
example₁ = mod-unif t₁
```

This evaluates to 17 :: [].

```
t₂ : T((ι ⇒ ι) ⇒ ι)
t₂ = Rec • t₁ • t₁
t₂-interpretation : ⟦ t₂ ⟧ ≡ λ α → rec α (α 17) (α 17)
t₂-interpretation = refl
example₂ example₂' : List ℕ
example₂ = mod-unif t₂
example₂' = mod-cont t₂ (λ i → i)
```

These examples evaluate to 17 :: 17 :: 17 :: 0 :: 1 :: [] and to a list whose members are all 17.

```
Add : T(ι ⇒ ι ⇒ ι)
Add = Rec · Succ
infixl 0 _+_
_+_ : ∀{ɣ} → T(ɣ ⇒ ι) → T(ɣ ⇒ ι) → T(ɣ ⇒ ι)
x + y = K · Add • x • y

t₃ : T((ι ⇒ ι) ⇒ ι)
t₃ = Rec • (v • Number 1) • (v • Number 2 + v • Number 3)
t₃-interpretation : ⟦ t₃ ⟧ ≡ λ α → rec α (α 1) (rec succ (α 2) (α 3))
t₃-interpretation = refl
example₃ example₃' : List ℕ
example₃ = mod-cont t₃ succ
example₃' = mod-unif t₃
```

These examples evaluate to 3 :: 2 :: 1 :: 2 :: 3 :: 4 :: 5 :: 6 :: 7 :: 8 :: [] and 3 :: 2 :: 1 :: 1 :: 0 :: 1 :: 2 :: 1 :: 0 :: 1 :: 1 :: 0 :: 0 :: 1 :: 1 :: 0 :: 1 :: [].

```
length : {X : Set} → List X → ℕ
length [] = 0
length (x :: s) = succ(length s)
max : ℕ → ℕ → ℕ
max 0 x = x
max x 0 = x
max (succ x) (succ y) = succ(max x y)
Max : List ℕ → ℕ
Max [] = 0
Max (x :: s) = max x (Max s)

t₄ : T((ι ⇒ ι) ⇒ ι)
t₄ = Rec • ((v • (v • Number 2)) + (v • Number 3)) • t₃
t₄-interpretation : ⟦ t₄ ⟧ ≡ λ α → rec α (rec succ (α (α 2)) (α 3)) (rec α (α 1) (rec succ (α 2) (α 3)))
t₄-interpretation = refl
example₄ example₄' : ℕ
example₄ = length(mod-unif t₄)
example₄' = Max(mod-unif t₄)
```

These examples evaluate to 215 and 3.

```
t₅ : T((ι ⇒ ι) ⇒ ι)
t₅ = Rec • (v • (v • t₂ + t₄)) • (v • Number 2)
t₅-explicitly : t₅ ≡    (S · (S · Rec · (S · I · (S · (S · (K · (Rec · Succ)) · (S · I · (S
    · (S · Rec · (S · I · (K · (number 17)))) · (S · I · (K · (number 17))))))
    · (S · (S · Rec · (S · (S · (K · (Rec · Succ)) · (S · I · (S · I · (K · (number 2)))))
    · (S · I · (K · (number 3))))) · (S · (S · Rec · (S · I · (K · (number 1))))
    · (S · (S · (K · (Rec · Succ)) · (S · I · (K · (number 2)))) · (S · I · (K
    · (number 3))))))))) · (S · I · (K · (number 2))))
t₅-explicitly = refl
t₅-interpretation : ⟦ t₅ ⟧ ≡ λ α → rec α (α(rec succ (α(rec α (α 17) (α 17)))
        (rec α (rec succ (α (α 2)) (α 3))
        (rec α (α 1) (rec succ (α 2) (α 3)))))) (α 2)
t₅-interpretation = refl
example₅ example₅' example₅'' : ℕ
example₅ = length(mod-unif t₅)
example₅' = Max(mod-unif t₅)
example₅'' = Max(mod-cont t₅ succ)
```

These examples evaluate to 15551, 17 and 57. All evaluations reported above are instantaneous, except this last set, which takes about a minute in a low-end netbook. Using Church encoding of dialogue trees produces a dramatic performance improvement [7], with an instantaneous answer in these examples, because Klesli extension and the functor don't need to walk through trees to be performed.

3 Informal, rigorous proof

We now provide a self-contained, informal, rigorous version of the formal proof given above, in a foundationally neutral exposition.

We work with the combinatory version of (the term language of) Gödel's system T. We have a ground type ι and a right-associative type formation operation $- \Rightarrow -$. Every term as a unique type. We have the constants

(i) $\mathtt{Zero}\colon \iota$.

(ii) $\mathtt{Succ}\colon \iota \Rightarrow \iota$.

(iii) $\mathtt{Rec}_\sigma\colon (\sigma \Rightarrow \sigma) \Rightarrow \sigma \Rightarrow \iota \Rightarrow \sigma$.

(iv) $\mathtt{K}_{\sigma,\tau}\colon \sigma \Rightarrow \tau \Rightarrow \sigma$.

(v) $\mathtt{S}_{\rho,\sigma,\tau}\colon (\rho \Rightarrow \sigma \Rightarrow \tau) \Rightarrow (\rho \Rightarrow \sigma) \Rightarrow \rho \Rightarrow \tau$.

We omit the subscripts when they can be uniquely inferred from the context. If $t\colon \sigma \Rightarrow \tau$ and $u\colon \tau$ are terms, then so is $tu\colon \tau$, with the convention that this application operation is left associative. Write T_σ for the set of terms of type σ.

In the *standard interpretation*, we map a type expression σ to a set $[\![\sigma]\!]$ and a term $t\colon \sigma$ to an element $[\![t]\!] \in [\![\sigma]\!]$. These interpretations are defined by induction as follows:

$$[\![\iota]\!] = \mathbb{N}, \qquad [\![\sigma \Rightarrow \tau]\!] = [\![\tau]\!]^{[\![\sigma]\!]} = ([\![\sigma]\!] \to [\![\tau]\!]) \text{ (set of all functions } [\![\sigma]\!] \to [\![\tau]\!]\text{)},$$
$$[\![\mathtt{Zero}]\!] = 0, \qquad [\![\mathtt{Succ}]\!]n = n+1, \qquad [\![\mathtt{Rec}]\!]fxn = f^n(x),$$
$$[\![\mathtt{K}]\!]xy = x, \qquad [\![\mathtt{S}]\!]fgx = fx(gx), \qquad [\![tu]\!] = [\![t]\!]([\![u]\!]).$$

For any given three sets X, Y, Z, the set $\mathrm{D}\,XYZ$ of *dialogue trees* is inductively defined as follows:

(i) A leaf labeled by an element $z \in Z$ is a dialogue tree, written ηz.

(ii) If $\phi\colon Y \to \mathrm{D}\,XYZ$ is a Y-indexed family of dialogue trees and $x \in X$, then the tree with root labeled by x and with one branch leading to the subtree ϕy for each $y \in Y$ is also a dialogue tree, written $\beta\phi x$.

Such trees are well founded, meaning that every path from the root to a leaf is finite. The above notation gives functions

$$\eta :\ Z \to \mathrm{D}\,XYZ,$$
$$\beta :\ (Y \to \mathrm{D}\,XYZ) \to X \to \mathrm{D}\,XYZ.$$

Dialogue trees describe "computations" of functions $f\colon Y^X \to Z$. Leaves give answers, and labels of internal nodes are queries to an "oracle" $\alpha \in Y^X$, the argument of the function f. For any dialogue tree $d \in \mathrm{D}\,XYZ$, we inductively define a function $f_d\colon Y^X \to Z$ by

$$f_{\eta z}(\alpha) = z, \qquad f_{\beta\phi x}(\alpha) = f_{\phi(\alpha x)}(\alpha).$$

The functions $Y^X \to Z$ that arise in this way are called *eloquent*. Notice that the oracle α is queried finitely many times in this computation, because a dialogue tree is well founded. Hence the function $f = f_d\colon Y^X \to Z$ satisfies

$$\forall \alpha \in Y^X \quad \exists \text{finite } S \subseteq X \quad \forall \alpha' \in Y^X, \alpha =_S \alpha' \implies f\alpha = f\alpha',$$

where $\alpha =_S \alpha'$ is a shorthand for $\forall x \in S, \alpha x = \alpha' x$. When $X = Y = Z = \mathbb{N}$, this amounts to continuity in the product topology of $\mathbb{N}^{\mathbb{N}}$ with $\mathbb{N}$ discrete, which gives the Baire space.

For Y finite and X, Z arbitrary, the dialogue tree is finitely branching and hence finite by well-foundedness (or directly by induction), and so the set of potential queries to the oracle is finite, so that, for any $f = f_d\colon Y^X \to Z$ with Y finite,

$$\exists \text{finite } S \subseteq X\ \forall \alpha, \alpha' \in Y^X, \quad \alpha =_S \alpha' \implies f\alpha = f\alpha'.$$

When $Y = 2 = \{0, 1\}$ and $X = Z = \mathbb{N}$, this amounts to (uniform) continuity in the product topology of $2^{\mathbb{N}}$ with 2 discrete, which gives the Cantor space.

Clearly, any $\mathbb{N}$-branching tree $d \in \mathrm{D}\,\mathbb{N}\mathbb{N}\mathbb{N}$ can be pruned to a 2-branching tree $d' \in \mathrm{D}\,\mathbb{N}2\mathbb{N}$ so that $f_{d'} : 2^{\mathbb{N}} \to \mathbb{N}$ is the restriction of $f_d\colon \mathbb{N}^{\mathbb{N}} \to \mathbb{N}$ from sequences to binary sequences. Hence if we show that every T-definable function $\mathbb{N}^{\mathbb{N}} \to \mathbb{N}$ is eloquent, we conclude that every T-definable function $\mathbb{N}^{\mathbb{N}} \to \mathbb{N}$ is continuous and its restriction to $2^{\mathbb{N}}$ is uniformly continuous. For this purpose, we consider an auxiliary model of system T.

Define $\mathrm{B}\,X = \mathrm{D}\,\mathbb{N}\mathbb{N}X$. We remark that although B is the object part of a monad, as discussed in the introduction, it is not necessary to know this for the purposes of this proof. The data given below do obey the required laws to get a monad, but the details are left to the interested reader.

For any function $f\colon X \to \mathrm{B}\,Y$, inductively define $f^\sharp\colon \mathrm{B}\,X \to \mathrm{B}\,Y$ by

$$f^\sharp(\eta x) = fx,$$
$$f^\sharp(\beta\phi i) = \beta(\lambda j.f^\sharp(\phi j))i.$$

This says that the tree $f^\sharp(d)$ is d with each leaf labeled by x replaced by the subtree fx, with no changes to the internal nodes. Given $f\colon X \to Y$, we define $f\colon \mathrm{B}\,X \to \mathrm{B}\,Y$ by

$$\mathrm{B}\,f = (\eta \circ f)^\sharp.$$

Hence $\mathrm{B}\,f(d)$ replaces each label x of a leaf of d by the label $f(x)$, and we have the naturality condition

$$\begin{tikzcd} \mathrm{B}\,X \arrow[r, "\mathrm{B}\,f"] & \mathrm{B}\,Y \\ X \arrow[u, "\eta"] \arrow[r, "f"'] & Y. \arrow[u, "\eta"'] \end{tikzcd}$$

For each $\alpha \in \mathbb{N}^{\mathbb{N}}$ and any set X, define a map $\mathrm{decode}_\alpha\colon \mathrm{B}\,X \to X$ by

$$\mathrm{decode}_\alpha(d) = f_d(\alpha).$$

Then, by definition, $\mathrm{decode}_\alpha(\eta x) = x$, and hence the naturality of η gives that of decode_α:

$$\begin{tikzcd} \mathrm{B}\,X \arrow[r, "\mathrm{B}\,f"] \arrow[d, "\mathrm{decode}_\alpha"'] & \mathrm{B}\,Y \arrow[d, "\mathrm{decode}_\alpha"] \\ X \arrow[r, "f"'] & Y. \end{tikzcd} \tag{1}$$

It is also easy to see, by induction on dialogue trees, that

$$\begin{tikzcd} \mathrm{B}\,X \arrow[rr, "f^\sharp"] \arrow[d, "\mathrm{decode}_\alpha"'] & & \mathrm{B}\,Y \arrow[d, "\mathrm{decode}_\alpha"] \\ X \arrow[r, "f"'] & \mathrm{B}\,Y \arrow[r, "\mathrm{decode}_\alpha"'] & Y. \end{tikzcd} \tag{2}$$

Now define

$$\mathrm{generic} : \ \mathrm{B}\,\mathbb{N} \to \mathrm{B}\,\mathbb{N}$$
$$\mathrm{generic} = (\beta\eta)^\sharp.$$

Because $\beta\colon (\mathbb{N} \to \mathrm{B}\,\mathbb{N}) \to \mathbb{N} \to \mathrm{B}\,\mathbb{N}$ and $\eta\colon \mathbb{N} \to \mathrm{B}\,\mathbb{N}$, the function generic is well defined. Its crucial property is that

$$\begin{tikzcd} \mathrm{B}\,\mathbb{N} \arrow[r, "\text{generic}"] \arrow[d, "\text{decode}_\alpha"'] & \mathrm{B}\,\mathbb{N} \arrow[d, "\text{decode}_\alpha"] \\ \mathbb{N} \arrow[r, "\alpha"'] & \mathbb{N}. \end{tikzcd} \tag{3}$$

The proof that

$$\text{decode}_\alpha(\text{generic}\, d) = \alpha(\text{decode}_\alpha\, d)$$

is straightforward by induction on d.

Now define the B-interpretation of types as follows:

$$\mathrm{B}\llbracket\iota\rrbracket = \mathrm{B}(\llbracket\iota\rrbracket) = \mathrm{B}\,\mathbb{N}, \qquad \mathrm{B}\llbracket\sigma \Rightarrow \tau\rrbracket = \mathrm{B}\llbracket\tau\rrbracket^{\mathrm{B}\llbracket\sigma\rrbracket}.$$

For any type σ and $f\colon X \to \mathrm{B}\llbracket\sigma\rrbracket$, define $f^\sharp\colon \mathrm{B}\,X \to \mathrm{B}\llbracket\sigma\rrbracket$ by induction on σ, where the base case $\sigma = \iota$ is given by the above definition, and the induction step $\sigma = (\rho \Rightarrow \tau)$ is given pointwise as

$$f^\sharp dy = (\lambda x.fxy)^\sharp d.$$

Notice that $f\colon X \to \mathrm{B}\llbracket\rho\rrbracket \to \mathrm{B}\llbracket\tau\rrbracket$ and $f^\sharp\colon\ \mathrm{B}\,X \to \mathrm{B}\llbracket\rho\rrbracket \to \mathrm{B}\llbracket\tau\rrbracket$.

Next extend system T with a new constant $\Omega\colon \iota \Rightarrow \iota$, a formal oracle, and define the B-interpretation of terms as follows:

$$\mathrm{B}\llbracket\Omega\rrbracket = \text{generic}, \quad \mathrm{B}\llbracket\mathtt{Zero}\rrbracket = \eta 0, \quad \mathrm{B}\llbracket\mathtt{Succ}\rrbracket = \mathrm{B}(\lambda n.n+1), \quad \mathrm{B}\llbracket\mathtt{Rec}\rrbracket fx = (\lambda n.f^n(x))^\sharp,$$
$$\mathrm{B}\llbracket\mathtt{K}\rrbracket xy = x, \qquad \mathrm{B}\llbracket\mathtt{S}\rrbracket fgx = fx(gx), \qquad \mathrm{B}\llbracket tu\rrbracket = \mathrm{B}\llbracket t\rrbracket(\mathrm{B}\llbracket u\rrbracket).$$

We also need to consider the standard interpretation of system T extended with the oracle Ω. We treat the oracle as a free variable, as hence the value of this free variable has to be provided to define the interpretation:

$$\llbracket\Omega\rrbracket\alpha = \alpha, \qquad \llbracket\mathtt{Zero}\rrbracket\alpha = 0, \qquad \llbracket\mathtt{Succ}\rrbracket\alpha n = n+1, \qquad \llbracket\mathtt{Rec}\rrbracket\alpha fxn = f^n(x),$$
$$\llbracket\mathtt{K}\rrbracket\alpha xy = x, \qquad \llbracket\mathtt{S}\rrbracket\alpha fgx = fx(gx), \qquad \llbracket tu\rrbracket\alpha = \llbracket t\rrbracket\alpha(\llbracket u\rrbracket\alpha).$$

We claim that for any term $t\colon \iota$,

$$\llbracket t\rrbracket\alpha = \text{decode}_\alpha(\mathrm{B}\llbracket t\rrbracket).$$

To prove this, we work with a logical relation R_σ between functions $\mathbb{N}^\mathbb{N} \to \llbracket\sigma\rrbracket$ and elements of $\mathrm{B}\llbracket\sigma\rrbracket$ by induction on σ. For any $n\colon \mathbb{N}^\mathbb{N} \to \mathbb{N}$ and $n' \in \mathrm{B}\,\mathbb{N}$, we define

$$R_\iota nn' \iff \forall\alpha, n\alpha = \text{decode}_\alpha\, n',$$

and, for any $f\colon \mathbb{N}^\mathbb{N} \to \llbracket\sigma\rrbracket \to \llbracket\tau\rrbracket$ and $f'\colon \mathrm{B}\llbracket\sigma\rrbracket \to \mathrm{B}\llbracket\tau\rrbracket$, we define

$$R_{\sigma\to\tau}ff' \iff \forall x\colon \mathbb{N}^\mathbb{N} \to \llbracket\sigma\rrbracket,\ \forall x' : \mathrm{B}\llbracket\sigma\rrbracket,\ R_\sigma xx' \to R_\tau(\lambda\alpha, f\alpha(x\alpha))(f'x').$$

We need a technical lemma for dealing with the dialogue interpretation of `Rec`:

Claim 3.1 *For all* $g\colon \mathbb{N} \to \mathbb{N}^{\mathbb{N}} \to \mathrm{B}[\![\sigma]\!]$ *and* $g'\colon \mathbb{N} \to \mathrm{B}[\![\sigma]\!]$*, if*

$$\forall k \in \mathbb{N},\ R_\sigma(gk)(g'k),$$

then $\forall n\colon \mathbb{N}^{\mathbb{N}} \to \mathbb{N},\ \forall n' \in \mathrm{B}\,\mathbb{N},\ R_\iota nn' \to R_\sigma(\lambda\alpha \to g(n\alpha)\alpha)(g'n')^\sharp$.

The proof is straightforward by induction on types, using diagram 2.

Claim 3.2 $R_\sigma\,[\![t]\!]\,(\mathrm{B}[\![t]\!])$ *for every term* $t\colon \sigma$.

The proof is by induction on terms, using diagram 3 for the term Ω, diagram 1 for the term `Succ`, and Claim 3.1 for the term `Rec`. The terms `K` and `S` are immediate but perhaps laborious, and the induction step, namely term application, is easy. This gives, in particular:

Claim 3.3 *For every term* $t\colon (\iota \Rightarrow \iota) \Rightarrow \iota$*, we have* $[\![t]\!]\alpha = \mathrm{decode}_\alpha(\mathrm{B}[\![t\Omega]\!])$.

It follows that every T-definable function $f\colon \mathbb{N}^{\mathbb{N}} \to \mathbb{N}$ is eloquent, with dialogue tree given by $\mathrm{B}[\![t\Omega]\!]$, where $t\colon (\iota \Rightarrow \iota) \Rightarrow \iota$ is any term denoting f, and hence continuous, with uniformly continuous restriction to $2^{\mathbb{N}}$.

4 Discussion, questions and conjectures

It may not be apparent from the informal proof of Section 3 that the argument is constructive, but Section 2 provides a constructive rendering in Martin-Löf type theory. We emphasize that our proof doesn't invoke the Fan Theorem [15,2] or any constructively contentious axiom.

We have deliberately chosen system T in its combinatory form as the simplest and most memorable non-trivial higher-type language to illustrate the essence of the technique proposed here. It is clearly routine (as well as interesting and useful) to apply the technique to a number of well-known extensions of the simply-typed lambda-calculus. But, for instance, at the time of writing, dependent types seem to require further thought, particularly in the presence of universes. Can one, e.g. (generalize and) apply the technique developed here to show that all MLTT definable functions $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ are continuous, and that their restrictions to $(\mathbb{N} \to 2)$ are uniformly continuous, in the main versions of (intensional) MLTT? More ambitiously, does the technique apply to Homotopy Type Theory [14]?

As pointed out by one of the anonymous referees, the syntactical techniques of [15] give more information: for any term t of type $(\iota \Rightarrow \iota) \Rightarrow \iota$ one can construct a term $m : (\iota \Rightarrow \iota) \Rightarrow \iota$ such that m internalizes the modulus of continuity of t. We adapted our technique to achieve this, as reported in [7], by working with Church encodings of dialogue trees defined within system T, and turning our semantical interpretation into a compositional translation of system T into itself. A corollary is that the dialogue trees of T-definable functions $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$, being themselves T-definable, have height smaller than ϵ_0.

References

[1] A. Bauer and M. Pretnar. Programming with algebraic effects and handlers. Submitted for publication, 2012.

[2] M.J. Beeson. *Foundations of Constructive Mathematics*. Springer, 1985.

[3] E. Bishop. *Foundations of constructive analysis*. McGraw-Hill Book Co., New York, 1967.

[4] A. Bove and P. Dybjer. Dependent types at work. *Proceedings of Language Engineering and Rigorous Software Development, LNCS*, 5520:57–99, 2009.

[5] T. Coquand and G. Jaber. A note on forcing and type theory. *Fundam. Inf.*, 100(1-4):43–52, January 2010.

[6] T. Coquand and G. Jaber. A computational interpretation of forcing in type theory. In *Epistemology versus Ontology*, pages 203–213. Springer, 2012.

[7] M.H. Escardó. Continuity of Gödel's system T definable functionals via effectful forcing. Agda proof at http://www.cs.bham.ac.uk/~mhe/dialogue/, July 2012.

[8] P. Hancock, D. Pattinson, and N. Ghani. Representations of stream processors using nested fixed points. In *Logical Methods in Computer Science*, page 2009.

[9] W. A. Howard. Ordinal analysis of terms of finite type. *The Journal of Symbolic Logic*, 45:493–504, 1980.

[10] S.C. Kleene. Recursive functionals and quantifiers of finite types I. *Trans. Amer. Math. Soc*, 91, 1959.

[11] J. Longley. When is a functional program not a functional program? In *Proceedings of Fourth ACM SIGPLAN International Conference on Functional Programming*, pages 1–7. ACM Press, 1999.

[12] P.Hancock and A. Setzer. Interactive programs in dependent type theory. In *CSL*, pages 317–331, 2000.

[13] G. Plotkin and J. Power. Algebraic operations and generic effects. *Applied Categorical Structures*, 11, 2003.

[14] The Univalent Foundations Program. Homotopy type theory: Univalent foundations of mathematics. Technical report, Institute for Advanced Study, 2013.

[15] A. S. Troelstra, editor. *Metamathematical investigation of intuitionistic arithmetic and analysis*. Lecture Notes in Mathematics, Vol. 344. Springer-Verlag, Berlin, 1973.

[16] C. Xu and M.H. Escardó. A constructive model of uniform continuity. To appear in TLCA, 2013.

A Connection Between Concurrency and Language Theory

Zoltán Ésik[1]

Dept. of Computer Science
University of Szeged
Szeged, Hungary

Abstract

We show that three fixed point structures equipped with (sequential) composition, a sum operation, and a fixed point operation share the same valid equations. These are the theories of (context-free) languages, (regular) tree languages, and simulation equivalence classes of (regular) synchronization trees (or processes). The results reveal a close relationship between classical language theory and process algebra.

Keywords: Fixed point operations, iteration theories, context-free languages, regular tree languages, synchronization trees, simulation equivalence

1 Introduction

Iteration theories [7] capture the equational properties of fixed point operations including the least fixed point operation over continuous or monotone functions over cpo's or complete lattices or in rational algebraic theories [31,38], the initial fixed point operation over continuous functors over categories with directed colimits [6], Elgot's (pointed) iterative theories [15], theories of trees and synchronization trees, and many other structures. Actually it was shown in [16,29] that an iteration theory arises whenever there are enough least pre-fixed or initial fixed points around.

It was argued in [7,9] that all natural cartesian fixed point models lead to iteration theories. Moreover, it was proved in [35] that essentially every nontrivial subclass of iteration theories obeying a natural condition satisfies exactly the equations of iteration theories.

But several models have an additional structure, such as an additive structure, which interacts with the cartesian operations and the fixed point operation in a nontrivial way. The relationship between the iteration theory structure

[1] This publication is supported by the European Union and co-funded by the European Social Fund. Project title: 'Telemedicine-focused research activities on the field of mathematics, informatics and medical sciences'. Project number: TMOP-4.2.2.A-11/1/KONV-2012-0073

and the additional operations has been the subject of several papers, including [1,8,12,19,11,17,20,21,27,28] and the recent [24]. In many cases, it was possible to capture this relationship by a finite number of equational (or sometimes quasi-equational) axioms. As a byproduct of these results, it was possible to give complete sets of equational axioms for various bisimulation and trace based process behaviors, rational power series and regular languages, regular tree languages, and many other models.

The theory of simulation equivalence classes of (regular) synchronization trees over a set of action symbols, equipped with the cartesian operations, the least fixed point operation *and sum*, has a finite equational axiomatization relatively to iteration theories [19]. Incidentally, the very same equations hold for continuous or monotone functions over complete lattices equipped with the least fixed point operation and the pointwise binary supremum operation as sum, or more generally, in all '(ω-)continuous idempotent grove theories'. In this paper, our main new contribution is that two more well-known classes of structures relevant to computer science are of this sort, the theories of (regular) tree languages and the theories of (context-free) languages (Theorem 3.1). In our argument, we will make use of a concrete characterization of the free ω-continuous idempotent grove theories, which is a result of independent interest (cf. Theorem 4.3). The facts proved in the paper reveal a close relationship between models of concurrency, automata and language theory, and models of denotational semantics.

The results of this paper can be formulated in several different formalism including 'μ-terms', 'letrec expressions', or cartesian categories. We have chosen the simple language of Lawvere theories, i.e., cartesian categories generated by a single object. The extension of the results to many-sorted theories is straightforward.

2 Theories

In any category, we write the composition $f \cdot g$ of morphisms $f : a \to b$ and $g : b \to c$ in diagrammatic order, and we let $\mathbf{1}_a$ denote the identity morphism $a \to a$. For an integer $n \geq 0$, we let $[n]$ denote the set $\{1, \ldots, n\}$. When $n = 0$, this set is empty.

A *(Lawvere) theory* [32,7] is a small category T whose objects are the nonnegative integers such that each object n is the n-fold coproduct of object 1 with itself. The hom-set of morphisms $n \to p$ of a theory T is denoted $T(n, p)$. We assume that every theory comes with distinguished coproduct injections $i_n : 1 \to n$, $i \in [n]$, $n \geq 0$. Thus, for any sequence of morphisms $f_1, \ldots, f_n : 1 \to p$, there is a unique morphism $f : n \to p$ with $i_n \cdot f = f_i$, for all $i \in [n]$. We denote this unique morphism f by $\langle f_1, \ldots, f_n \rangle$ and call it the *tupling* of the f_i. When $n = 0$, we also write 0_p. Since 0 is initial object, 0_p is the unique morphism $0 \to p$. It is clear that $\mathbf{1}_n = \langle 1_n, \ldots, n_n \rangle$ for all $n \geq 0$. We require that $\mathbf{1}_1 = 1_1$, so that $\langle f \rangle = f$ for all $f : 1 \to p$. Since the object $n + m$ is the coproduct of objects n and m with respect to the coproduct injections

$$\begin{aligned}\kappa_{n,n+m} &= \langle 1_{n+m}, \ldots, n_{n+m} \rangle : n \to n + m\\ \lambda_{m,n+m} &= \langle (n + 1)_{n+m}, \ldots, (n + m)_{n+m} \rangle : m \to n + m,\end{aligned}$$

each theory is equipped with a *pairing* operation mapping a pair of morphisms (f, g)

with $f : n \to p$ and $g : m \to p$ to $\langle f, g\rangle : n + m \to p$:

$$\begin{tikzcd}
n \arrow[r, "\kappa"] \arrow[dr, "f"'] & n+m \arrow[d, "\langle f,g\rangle"] & m \arrow[l, "\lambda"'] \arrow[dl, "g"] \\
& p &
\end{tikzcd}$$

The pairing operation is associative and satisfies $\langle f, 0_p\rangle = f = \langle 0_p, f\rangle$ for all $f : n \to p$.

Also, we can define for $f : n \to p$ and $g : m \to q$ the morphism $f \oplus g : n + m \to p + q$ as $\langle f \cdot \kappa_{p,p+q},\, g \cdot \lambda_{q,p+q}\rangle$. Then $f \oplus g$ is the unique morphism $n + m \to p + q$ with

$$\begin{aligned}
\kappa_{n,n+m} \cdot (f \oplus g) &= f \cdot \kappa_{p,p+q}\\
\lambda_{m,n+m} \cdot (f \oplus g) &= g \cdot \lambda_{q,p+q}.
\end{aligned}$$

$$\begin{tikzcd}
n \arrow[r, "\kappa"] \arrow[d, "f"'] & n+m \arrow[d, "f\oplus g"] & m \arrow[l, "\lambda"'] \arrow[d, "g"] \\
p \arrow[r, "\kappa"'] & p+q & q \arrow[l, "\lambda"]
\end{tikzcd}$$

The $\oplus$ operation is associative, and $0_0 \oplus f = f = f \oplus 0_0$ for all $f : n \to p$. Also,

$$(f \oplus g) \cdot \langle h, k\rangle = \langle f \cdot h,\, g \cdot k\rangle$$

for all $f : n \to p$, $g : m \to q$, $h : p \to r$ and $k : q \to r$.

Each theory T may be seen as a many-sorted algebra, whose set of sorts is the set $\mathbb{N} \times \mathbb{N}$ of all ordered pairs of nonnegative integers, satisfying certain equational axioms, see e.g. [7]. Morphisms of theories are functors preserving objects and distinguished morphisms. It follows that any theory morphism preserves the tupling, (and pairing) operations. The kernels of theory morphisms are called *theory congruences*. The *quotient* $T/\equiv$ of a theory T with respect to a theory congruence is defined as usual. A *subtheory* of a theory T is a theory T' whose set of morphisms is included in the morphisms of T such that the natural embedding of T' into T is a theory morphism $T' \to T$. See [7] for more details.

We end this section by providing some examples.

Let $X = \{x_1, x_2, \ldots\}$ denote a fixed countably infinite set of variables, and let A be a set disjoint from X. For each $p \geq 0$, let $X_p = \{x_1, \ldots, x_p\}$. The theory $\mathbf{W}_A$ has as morphisms $1 \to p$ all words in $(A \cup X_p)^*$. A morphism $n \to p$ is an n-tuple of morphisms $1 \to p$. For morphisms $u = (u_1, \ldots, u_n) : n \to p$ and $v = (v_1, \ldots, v_p) : p \to q$, we define $u \cdot v = (u_1 \cdot v, \ldots, u_n \cdot v)$, where for each $i \in [n]$, $u_i \cdot v$ is the word obtained from u_i by substituting a copy of v_j for each occurrence of the variable x_j in v_i, for all $j \in [p]$. Equipped with this composition operation and the morphisms $\mathbf{1}_n = (x_1, \ldots, x_n) : n \to n$ as identity morphisms, $\mathbf{W}_A$ is a category. In fact, $\mathbf{W}_A$ is a theory with distinguished morphisms $i_n = x_i : 1 \to n$, $i \in [n]$, $n \geq 0$.

Suppose now that $\Sigma = \biguplus_{k\geq 0} \Sigma_k$ is a *ranked set* which is disjoint from X. We may view Σ as a pure set and form the theory $\mathbf{W}_\Sigma$. Consider the subtheory $\mathbf{Tree}_\Sigma$

of $\mathbf{W}_\Sigma$ consisting of the Σ-trees (or Σ-terms). A morphism $1 \to p$ in $\mathbf{Tree}_\Sigma$ is a well-formed word in $(\Sigma \cup X_p)^*$ which is either a variable in X_p or a word of the form $\sigma t_1 \dots t_k$ for a letter $\sigma \in \Sigma_k$ and trees $t_1, \dots, t_k : 1 \to p$. A morphism $n \to p$ is an n-tuple of morphisms $n \to p$. It is well-known that the theory $\mathbf{Tree}_\Sigma$ is the free theory, freely generated by Σ. Indeed, each letter $\sigma \in \Sigma_n$ may be identified with a tree in $\mathbf{Tree}_\Sigma(1,n)$ so that given any theory T and rank preserving function $\varphi : \Sigma \to T$, there is a unique theory morphism $\varphi^\sharp : \mathbf{Tree}_\Sigma \to T$ extending φ.

Remark 2.1 *If in the previous example Σ is empty, then we obtain the initial theory Θ. A morphism $n \to p$ of this initial theory is a tupling of distinguished morphisms and may be identified with a function $[n] \to [p]$, so that composition corresponds to composition of functions. A* base morphism *of a theory T is a morphism that arises as the image of a morphism in the initial theory with respect to the unique theory morphism $\Theta \to T$. For example, the base morphisms $n \to p$ in a theory $\mathbf{W}_A$ are the morphisms of the form $(x_{1\rho}, \dots, x_{n\rho})$, where ρ is a function $[n] \to [p]$. In any nontrivial theory, we may represent base morphisms $n \to p$ as functions $[n] \to [p]$.*

3 Statement of the main result

By taking sets of morphisms of a theory T, we may *sometimes* define a new theory $P(T)$. (For a more general construction, the reader is referred to [10].) The morphisms $1 \to p$ in $P(T)$ are all sets $L \subseteq T(1,p)$. A morphism $n \to p$ is an n-tuple $(L_1, \dots, L_n)$ of morphisms $1 \to p$, including the tuple $0_{n,p} = (\emptyset, \dots, \emptyset)$. To define composition, suppose that $L : 1 \to p$ and $K = (K_1, \dots, K_p) : p \to q$. Then we define $L \cdot K : 1 \to q$ to be the set of all morphisms $1 \to q$ in T of the form

$$f \cdot \langle g_1, \dots, g_m \rangle$$

such that $f : 1 \to m$ in T and there is a base morphism $\rho : m \to p$ with $f \cdot \rho \in L$ and $g_i \in K_{i\rho}$ for all $i \in [m]$. When $L = (L_1, \dots, L_n) : n \to p$, we define $L \cdot K$ as the morphism $(L_1 \cdot K, \dots, L_n \cdot K) : n \to q$. For each $n \geq 0$, the identity morphism $\mathbf{1}_n$ is the morphism $(\{1_n\}, \dots, \{n_n\}) : n \to n$, and the ith distinguished morphism $1 \to n$ is $\{i_n\}$. When $P(T)$ is a theory, we call it a *power-set theory.*

Suppose that $P(T)$ is a power-set theory. We may also equip $P(T)$ with a sum operation, denoted $+$ and defined by component-wise set union. We define

$$L + L' = (L_1 \cup L'_1, \dots, L_n \cup L'_n) : n \to p,$$

for all $L = (L_1, \dots, L_n) : n \to p$ and $L' = (L'_1, \dots, L'_n) : n \to p$ in $P(T)$. It is clear that, equipped with the operation $+$ and the constant $0_{n,p}$, each hom-set $P(T)(n,p)$ is a commutative, idempotent monoid. Moreover,

$$i_n \cdot (L + L') = i_n \cdot L + i_n \cdot L' \tag{1}$$

$$i_n \cdot 0_{n,p} = 0_{1,p} \tag{2}$$

$$(L + L') \cdot K = L \cdot K + L' \cdot K \tag{3}$$

$$0_{n,p} \cdot K = 0_{n,q}, \tag{4}$$

for all $L, L' : n \to p$ and $K : p \to q$. (Here, we adapt the convention that composition has higher precedence than sum.) Thus, $P(T)$ is an *idempotent grove theory*, cf. [7] or Section 4.

The above power-set construction is applicable to the theories $\mathbf{W}_A$ and $\mathbf{Tree}_\Sigma$, yielding the idempotent grove theories $\mathbf{Lang}_A$ of languages over A and $\mathbf{TreeLang}_\Sigma$ of tree languages over Σ. In $\mathbf{Lang}_A$, composition is the usual operation of 'language substitution'. In $\mathbf{TreeLang}_\Sigma$, it corresponds to the 'OI-substitution' of [14].

Any power-set theory $P(T)$ is naturally equipped with a partial order $\subseteq$ defined by component-wise set inclusion. It is clear that each hom-set $P(T)(n,p)$ is a complete lattice with least element $0_{n,p}$, moreover, the theory operations are monotone, in fact continuous. (Composition preserves all suprema in its first argument, and tupling preserves all suprema in each of its arguments.) Thus, we can define a *dagger operation* $^\dagger : T(P)(n, n+p) \to T(P)(n,p)$ $(n, p \geq 0)$, $L \mapsto L^\dagger$, by taking the least solution of the fixed point equation $X = L \cdot \langle X, \mathbf{1}_p \rangle$. In particular, the theories $\mathbf{Lang}_A$ and $\mathbf{TreeLang}_\Sigma$ are also equipped with a dagger operation. The least subtheory of $\mathbf{Lang}_A$ containing the finite languages which is closed under dagger is the theory $\mathbf{CFL}_A$ of context-free languages, and the least subtheory of $\mathbf{TreeLang}_A$ containing the finite tree languages which is closed under dagger is the theory $\mathbf{Reg}_\Sigma$ of regular tree languages [17,21,30]. Both $\mathbf{CFL}_A$ and $\mathbf{Reg}_\Sigma$ are idempotent grove theories.

We define yet another class of theories equipped with both an additive structure and a dagger operation, the theories of simulation equivalence classes of synchronization trees. A *hyper-tree* consists of a countable set V of vertices and a countable set E of edges, each edge e having a source v in V and an ordered sequence of target vertices $(v_1, \ldots, v_n) \in V^n$, for some $n \geq 0$. There is a distinguished vertex, the *root* v_0, such that each vertex v is the target vertex of a unique path from v_0 to v. An isomorphism between hyper-trees is determined by a bijection between the vertices and a bijection between the edges that jointly preserve the root and the source and target of the edges.

Synchronization trees over a set A of *action symbols* were defined in [37]. A (slight) generalization of synchronization trees for ranked sets is given in [19]. Suppose that Σ is a ranked set. A *synchronization tree* $t = (V_t, E_t, \lambda_t) : 1 \to p$ over Σ is a hyper-tree with vertex set V_t, hyper-edges E_t, equipped with a labeling function $\lambda_t : E_t \to \Sigma \cup \{\mathsf{ex}_1, \ldots, \mathsf{ex}_p\}$, where the ex_i are referred to as the *exit symbols*. Each hyper-edge $e : v \to (v_1, \ldots, v_n)$ with source v and target $(v_1, \ldots, v_n)$ is labeled in Σ_n, when $n \geq 1$, or by an exit symbol or a symbol in Σ_0, when $n = 0$. When t is a synchronization tree and v is a vertex of t, then the vertices 'accessible' from v (including v) span the *subtree* $t|_v$. The edges of $t|_v$ are those edges of t having a source accessible from v. An isomorphism between synchronization trees is an isomorphism of the underlying hyper-trees which preserves the labeling. We usually identify isomorphic synchronization trees. A synchronization tree $n \to p$ over Σ is an n-tuple $(t_1, \ldots, t_n)$ of synchronization trees $1 \to p$ over Σ. A synchronization tree $t : 1 \to p$ is *finite* if its set of edges is finite (and thus its vertex set is also finite), *finitely branching* if each vertex is the source of a finite number of edges, and *regular*, if it has a finite number of subtrees (up to isomorphism) and only a finite number of letters from Σ appear as edge labels. A synchronization tree $t : n \to p$

is finite (finitely branching, regular, resp.) if its components are all finite (finitely branching, regular, resp.).

We may identify each letter $\sigma \in \Sigma_n$ with the finite synchronization tree $1 \to n$ having and edge $v_0 \to (v_1, \ldots, v_n)$ labeled σ, where v_0 is the root, and an edge originating in v_i labeled ex_i for each $i \in [n]$. In the same way, we may view each exit symbol ex_i as a tree $1 \to n$ for each $i \in [n]$, $n \geq 0$.

Synchronization trees over Σ form a theory $\mathbf{ST}_\Sigma$. When $t : 1 \to p$ and $t' = (t'_1, \ldots, t'_p) : p \to q$, then $t \cdot t' : 1 \to q$ is constructed from t by replacing each edge of t labeled ex_i for some $i \in [p]$ by a copy of t'_i. When $t = (t_1, \ldots, t_n) : n \to p$, then $t \cdot t' = (t_1 \cdot t', \ldots, t_n \cdot t') : n \to q$. For each $i \in [n]$, the distinguished morphism i_n is the tree having a single edge labeled ex_i. For synchronization trees $t, t' : 1 \to p$, we also define $t + t' : 1 \to p$ as the tree obtained from (disjoint copies of) t and t' by merging the roots. When $t = (t_1, \ldots, t_n) : n \to p$ and $t' = (t'_1, \ldots, t'_n) : n \to p$, then $t + t' = (t_1 + t'_1, \ldots, t_n + t'_n) : n \to p$. We also define $0_{1,p}$ as the tree $1 \to p$ having no edge, and $0_{n,p} = (0_{1,p}, \ldots, 0_{1,p}) : n \to p$, for all $n, p \geq 0$. Clearly, each hom-set of $\mathbf{ST}_\Sigma$ is a commutative monoid and (1)–(4) hold, so that $\mathbf{ST}_\Sigma$ is a *grove theory* [7]. We also define the grove theories $\mathbf{FST}_\Sigma$ of finite and $\mathbf{RST}_\Sigma$ of regular synchronization trees over Σ.

Suppose that t and t' are synchronization trees $1 \to p$ over Σ. A *simulation* [33,34] $t \to t'$ is a relation $R \subseteq V_t \times V_{t'}$, relating the roots such that whenever $e : v \to (v_1, \ldots, v_n)$ is an edge of t and vRv', then there is an equally labeled edge $v' \to (v'_1, \ldots, v'_n)$ of t' such that $v_i R v'_i$ for all i. Note that the domain of a simulation $R : t \to t'$ is V_t. It is well-known that simulations compose, so that if $t, t', t'' : 1 \to p$ and R is a simulation $t \to t'$ and R' is a simulation $t' \to t''$, then the relational composition of R and R' is a simulation $t \to t''$. When $t = (t_1, \ldots, t_n)$ and $t' = (t'_1, \ldots, t'_n)$ are synchronization trees $n \to p$, a simulation $t \to t'$ is an n-tuple $(R_1, \ldots, R_n)$, where each R_i is a simulation $t_i \to t'_i$. We say that t and t' are *simulation equivalent*, denoted $t \equiv_s t'$, if there are simulations $t \to t'$ and $t' \to t$. The relation $\equiv_s$ is a *grove theory congruence* of $\mathbf{ST}_\Sigma$, i.e., a theory congruence which preserves the sum operation, giving rise to the grove theory $\mathbf{SST}_\Sigma = \mathbf{ST}_\Sigma / \equiv_s$. We will denote the simulation equivalence class of a tree t by $[t]_s$, or sometimes just $[t]$. Moreover, when $t = (t_1, \ldots, t_n) : n \to p$, we identify $[t]_s$ with $([t_1]_s, \ldots, [t_n]_s)$.

We define the relation $t \sqsubseteq_s t'$ for synchronization trees $t, t' : n \to p$ iff there is a simulation $t \to t'$. Also, we define $[t]_s \sqsubseteq_s [t']_s$ iff $t \sqsubseteq_s t'$, since the definition is independent of the choice of the representatives of the equivalence classes. Since simulations compose, the relation $\sqsubseteq_s$ is a pre-order on synchronization trees and a partial order on simulation equivalence classes. Each hom-set of $\mathbf{SST}_\Sigma$ has all countable suprema. Indeed, when t_i, $i \in I$, is a countable family of trees $n \to p$, then $\sup_{i \in I} [t_i]_s = [t]_s$ for the tree $t = \sum_{i \in I} t_i : n \to p$ obtained by taking the disjoint union of the t_i and identifying the roots. When I is empty, the sum is the tree $0_{n,p}$. The theory operations are ω-continuous, so that we can define a dagger operation. For each $f = [t]_s : n \to n + p$ in $\mathbf{SST}_\Sigma$, $f^\dagger : n \to p$ is the least solution of the fixed-point equation $x = f \cdot \langle x, \mathbf{1}_p \rangle$. The least subtheory of $\mathbf{SST}_\Sigma$ containing the finite synchronization trees which is closed under dagger is the theory $\mathbf{SRST}_\Sigma$ of simulation equivalence classes containing at least one regular tree. Further, we denote by $\mathbf{SFST}_\Sigma$ the subtheory determined by those simulation equivalence classes containing at least one finite synchronization tree. Both $\mathbf{SRST}_\Sigma$ and $\mathbf{SFST}_\Sigma$ are closed under the sum operation, and both of them are grove theories. Note that we may identify $\mathbf{SRST}_\Sigma$ with $\mathbf{RST}_\Sigma/{\equiv_s}$ and $\mathbf{SFST}_\Sigma$ with $\mathbf{FST}_\Sigma/{\equiv_s}$.

A *term* is a well-formed expression composed of morphism variables and constants for the distinguished morphisms using the theory operations, sum, and dagger. Each term has a source n and a target p, for some nonnegative integers n, p.

We are now ready to state our main result. We may view each set A as a ranked set where each letter has rank 1.

Theorem 3.1 *The following conditions are equivalent for terms $t, t' : n \to p$.*

(i) The identity $t = t'$ holds in all power-set theories $P(T)$, where T is a theory.

(ii) The identity $t = t'$ holds in all theories $\mathbf{Lang}_A$ (or $\mathbf{CFL}_A$), where A is a set.

(iii) The identity $t = t'$ holds in all theories $\mathbf{TreeLang}_\Sigma$ (or $\mathbf{Reg}_\Sigma$), where Σ is a ranked set.

(iv) The identity $t = t'$ holds in all theories $\mathbf{SST}_\Sigma$ (or $\mathbf{SRST}_\Sigma$), where Σ is a ranked set.

(v) The identity $t = t'$ holds in all theories $\mathbf{SST}_A$ (or $\mathbf{SRST}_A$), where A is a set.

(In (ii) and (v), we could as well require that A is a two-element set.) The proof of Theorem 3.1 will be completed in Section 5.

Since simulation equivalence is known to be decidable (in polynomial time for finite process graphs, cf. [2,36]), it follows that it is decidable for terms $t, t' : n \to p$ whether $t = t'$ holds in all theories $\mathbf{CFL}_A$. This fact is in contrast with the well-known undecidability of the equivalence problem for context-free grammars. Intuitively, our positive result is due to the fact that we are interested in the equivalence of terms under *all* possible interpretations of the morphism variables as context-free languages. By restricting the interpretations to those mapping a fixed morphism variable $1 \to 2$ to the language $\{x_1 x_2\}$ (or by adding to our operations a constant for this language), we would run into undecidability.

Remark 3.2 Languages and tree languages satisfy

$$L \cdot \langle L_1 + L_1', \ldots, L_n + L_n' \rangle = \sum_{K_i \in \{L_i, L_i'\}} L \cdot \langle K_1, \ldots, K_n \rangle$$

$$L \cdot \langle L_1, \ldots, 0_{1,p}, \ldots, L_n \rangle = 0_{1,p}$$

for all $L : 1 \to n$, and $L_i, L_i' : 1 \to p$ whenever each of the variables $x_1, \ldots, x_n$ occurs *exactly once* in each word/tree of L. However, these equations do not hold universally.

4 Free ω-continuous idempotent grove theories

Most proofs in this section are removed due to space limitations. They can be found in the full version [23].

Recall from [7] that a grove theory is a theory T with a commutative additive monoid structure $(T(n,p), +, 0_{n,p})$ on each hom-set such that (1)–(4) hold. An idempotent grove theory is a grove theory with an idempotent sum operation. A morphism of (idempotent) grove theories is a theory morphism preserving + and the constants $0_{n,p}$. When T is an idempotent grove theory, we may define a partial order $\leq$ on each hom-set $T(n,p)$ by $f \leq g$ iff $f + g = g$. It is clear that $0_{n,p}$ is the least element of $T(n,p)$ with respect to this partial order, and the tupling and sum operations preserve the order. Composition necessarily preserves the order in the first argument, but not necessarily in the second. When it does, we call T an *ordered idempotent grove theory.* Moreover, when $f, g : n \to p$, then $f \leq g$ iff $i_n \cdot f \leq i_n \cdot g$ for all $i \in [n]$. Thus, the partial order on morphisms $n \to p$ is determined by the order on the morphisms $1 \to p$. Morphisms of idempotent grove theories necessarily preserve the order.

We say that an idempotent grove theory is *ω-continuous* if the supremum $\sup_k f_k$ of each ω-chain $(f_k : n \to p)_k$ exists and composition preserves the supremum of ω-chains in both arguments. It follows that every ω-continuous idempotent grove theory is ordered, and the supremum of every countable family of morphisms $f_i : n \to p$, $i \in I$ exists. Moreover, composition preserves the supremum of all countable families in its first argument. A morphism of ω-continuous idempotent grove theories preserves the supremum of ω-chains.

Examples of ω-continuous idempotent grove theories include the theories $\mathbf{Lang}_A$, $\mathbf{TreeLang}_\Sigma$ and $\mathbf{SST}_\Sigma$ defined above. In $\mathbf{Lang}_A$ and $\mathbf{TreeLang}_\Sigma$, the relation $\leq$ is the component-wise set inclusion relation $\subseteq$, whereas it is the relation $\sqsubseteq_s$ in $\mathbf{SST}_A$. Each of these theories is equipped with a dagger operation. More generally, we may define a dagger operation in any ω-continuous idempotent grove theory: for a morphism $f : n \to n + p$, $f^\dagger : n \to p$ is the least solution of the equation $x = f \cdot \langle x, \mathbf{1}_p \rangle$ in the variable $x : n \to p$. We have $f^\dagger = \sup_k f^{(k)}$, where $f^{(0)} = 0_{n,p}$ and $f^{(k+1)} = f \cdot \langle f^{(k)}, \mathbf{1}_p \rangle$, for all $k \geq 0$. It is clear that every morphism of ω-continuous idempotent grove theories preserves dagger.

An *ideal* in $\mathbf{FST}_\Sigma(n,p)$ is a nonempty set $Q \subseteq \mathbf{FST}_\Sigma$ which is downward closed with respect to the relation $\sqsubseteq_s$. An *ω-ideal* is an ideal Q which is generated by some ω-chain $(t_k)_k$ of trees $t_k : n \to p$ in $\mathbf{FST}_\Sigma$ with $t_k \sqsubseteq_s t_{k+1}$ for all $k \geq 0$. Note that we may identify any (ω)-ideal $Q \subseteq \mathbf{FST}_\Sigma(n,p)$ with an n-tuple of (ω)-ideals $(Q_1, \ldots, Q_n)$, where $Q_i \subseteq \mathbf{FST}_\Sigma(1,p)$ is the set of all ith components of the members of Q, for each $i \in [n]$. We may recover Q from $(Q_1, \ldots, Q_n)$ as the set $\{t : n \to p : i_n \cdot t \in Q_i \text{ for all } i \in [n]\}$.

We may turn ω-ideals into an idempotent grove theory $\omega\mathbf{SFST}_\Sigma$. The set of morphisms $n \to p$ in $\omega\mathbf{SFST}_\Sigma$ is the collection of all ω-ideals $Q \subseteq \mathbf{FST}_\Sigma(n,p)$. When $Q : n \to p$ and $Q' : p \to q$, then we define $Q \cdot Q' : n \to q$ to be the ideal generated by the set of all trees $f \cdot g$ with $f : n \to p$ in Q and $g : p \to q$ in Q'. When Q and Q' are generated by the ω-chains $(f_k)_k$ and $(g_k)_k$, then $Q \cdot Q'$ is the ω-ideal generated by $(f_k \cdot g_k)_k$. For each $i \in [n]$, $n \geq 0$, the distinguished morphism $1 \to n$ is the ideal generated by the tree ex_i. The sum $Q + Q' : n \to p$ of $Q : n \to p$ and $Q' : n \to p$ is defined as the ideal generated by $\{f + g : f \in Q,\ g \in Q'\}$. It is easy to see that this is again an ω-ideal. The morphism $0_{n,p} : n \to p$ in $\omega\mathbf{SFST}_\Sigma$ is the ideal containing only the tree $0_{n,p}$.

There is a canonical embedding of $\mathbf{SFST}_\Sigma$ into $\omega\mathbf{SFST}_\Sigma$ which maps the simulation equivalence class of a finite tree $t : n \to p$ to the principal ω-ideal $\{t' : n \to p : t' \sqsubseteq_s t\}$. It is easy to see that this defines an (ordered) idempotent grove theory morphism $\mathbf{SFST}_\Sigma \to \omega\mathbf{SFST}_\Sigma$.

An ω-ideal in $\mathbf{SFST}_\Sigma(n,p)$ is defined in the same way as in $\mathbf{FST}_\Sigma(n,p)$ using the partial order $\sqsubseteq_s$. We may identify any ω-ideal $Q \subseteq \mathbf{SFST}_\Sigma(n,p)$ with an ω-ideal in $Q' \subseteq \mathbf{FST}_\Sigma(n,p)$ which is the union of all simulation equivalence classes of the trees in Q. Using this identification, $\omega\mathbf{SFST}_\Sigma$ is just the completion of $\mathbf{SFST}_\Sigma$ by ω-ideals as defined in [5] [2] It follows from the main result of [5] that $\omega\mathbf{SFST}_\Sigma$ is an ω-continuous idempotent grove theory, and we thus have:

Proposition 4.1 *The theory $\omega\mathbf{SFST}_\Sigma$ is the free ω-continuous idempotent grove theory, freely generated by $\mathbf{SFST}_\Sigma$. Given any ω-continuous idempotent grove theory T and an ordered idempotent grove theory morphism $\varphi : \mathbf{SFST}_\Sigma \to T$, there is a unique ω-idempotent grove theory morphism $\varphi^\sharp : \omega\mathbf{SFST}_\Sigma \to T$ extending φ.*

Proposition 4.2 *The theory $\mathbf{SFST}_\Sigma$ is the free ordered idempotent grove theory, freely generated by Σ.*

For a proof, see [23]. By Proposition 4.2 and Proposition 4.1, we immediately have:

Theorem 4.3 *For each ranked alphabet Σ, the theory $\omega\mathbf{SFST}_\Sigma$ is the free ω-continuous idempotent grove theory, freely generated by Σ.*

Our next task is to relate ω-ideals of finite synchronization trees to possibly infinite synchronization trees.

For each tree $t : n \to p$ in $\mathbf{ST}_\Sigma$, let $K(t)$ denote the set of all *finite* trees $t' : n \to p$ with $t' \sqsubseteq_s t$.

Proposition 4.4 *A set of finite trees $Q \subseteq \mathbf{FST}_\Sigma(n,p)$ is an ω-ideal iff $Q = K(t)$ for some (possibly infinite) tree $t : n \to p$ in $\mathbf{ST}_\Sigma$.*

Proposition 4.5 *Suppose that $t, t' : n \to p$ in $\mathbf{ST}_\Sigma$. If $t \sqsubseteq_s t'$ then $K(t) \subseteq K(t')$. Moreover, if t and t' are finitely branching, or simulation equivalent to some finitely branching trees, and if $K(t) \subseteq K(t')$, then $t \sqsubseteq_s t$.*

For proofs of the above facts, see [23].

Corollary 4.6 *If $t, t' : n \to p$ in $\mathbf{ST}_\Sigma$ are simulation equivalent to finitely branching trees, then $t \sqsubseteq_s t'$ iff $K(t) \subseteq K(t')$, and $t \equiv_s t'$ iff $K(t) = K(t')$.*

Example 4.7 *Let t be the infinitely branching tree $t = \sum_{n \geq 0} \sigma^n \cdot 0_{1,0} : 1 \to 0$, and let $t' = \sigma^\omega : 1 \to 0$, a tree consisting of a single infinite branch with edges labeled $\sigma \in \Sigma_1$. Then $K(t) = K(t')$ but $t \equiv_s t'$ does not hold.*

Since every regular synchronization tree is simulation equivalent to a finitely branching regular tree, we have:

Corollary 4.8 *Suppose that $t, t' : n \to p$ in $\mathbf{RST}_\Sigma$. Then $t \sqsubseteq_s t'$ iff $K(t) \subseteq K(t')$ and $t \equiv_s t'$ iff $K(t) = K(t')$.*

From Theorem 4.3 and Corollary 4.8, we obtain:

Corollary 4.9 *Suppose that Σ is a ranked set, T is an ω-continuous idempotent grove theory and $\varphi : \Sigma \to T$ is a rank preserving function. Then there is a unique way to extend φ to an idempotent grove theory morphism $\varphi^\sharp : \mathbf{SRST}_\Sigma \to T$ pre-*

[2] Actually [5] uses a different representation of ω-ideals.

serving dagger.

Proof. Suppose that T is an ω-continuous idempotent grove theory and φ is a rank preserving function $\Sigma \to T$. We may extend φ to a morphism $\psi : \omega\mathbf{SFST}_\Sigma \to T$ of ω-continuous idempotent grove theories. We know that $\mathbf{SRST}_\Sigma$ embeds in $\omega\mathbf{SFST}_\Sigma$ by the function which maps a regular tree $t : n \to p$ to $K(t)$. It is a routine matter to verify that the embedding preserves the theory operations, the additive structure, and dagger. Thus, we may identify $\mathbf{SRST}_\Sigma$ with a subtheory of $\omega\mathbf{SFST}_\Sigma$. The restriction of ψ to $\mathbf{SRST}_\Sigma$ is the required extension $\varphi^\sharp : \mathbf{SRST}_\Sigma \to T$. □

Remark 4.10 Corollary 4.9 is also derivable from a stronger result in [19], where it is shown (using the language of μ-terms) that simulation equivalence classes of regular synchronization trees form the free theories in a class of iteration theories with an additive structure satisfying certain axioms. Our aim here was to derive this result from Theorem 4.3.

5 Proof of the main result

In this section our aim is to prove Theorem 3.1.

Recall that we may view each set A as a ranked set of letters of rank 1. We start by showing that for each ranked set Σ there is some set A such that $\mathbf{SST}_\Sigma$ embeds in $\mathbf{SST}_A$ and $\mathbf{SRST}_\Sigma$ embeds in $\mathbf{SRST}_A$.

Proposition 5.1 *For every ranked set Σ there exists a set A and an injective (idempotent) grove theory morphism $\mathbf{SRST}_\Sigma \to \mathbf{SRST}_A$ preserving dagger.*

Proof. When Σ is a ranked set, define $A = \overline{\Sigma} \cup \{\#\}$, where $\overline{\Sigma} = \{\overline{\sigma} : \sigma \in \Sigma\}$. Our aim is to show that there is an injective dagger preserving grove theory morphism $\mathbf{SRST}_\Sigma \to \mathbf{SRST}_A$.

Consider the function φ which maps the simulation equivalence class of the tree corresponding to a letter $\sigma \in \Sigma_n$, $n \geq 0$, to the simulation equivalence class of the synchronization tree

$$s_\sigma = \overline{\sigma} \cdot (\# \cdot 1_n + \#^2 \cdot 2_n + \ldots + \#^n \cdot n_n) : 1 \to n$$

in $\mathbf{ST}_A$. (Recall that $\overline{\sigma}$ has rank 1. The tree s_σ has a single edge originating in the root, which is labeled $\overline{\sigma}$. The target of this edge is the source of n branches, a branch of the form $\#^i \cdot i_n$ for each $i \in [n]$.) When $n = 0$, the simulation equivalence class of the tree σ is mapped to the equivalence class $[\overline{\sigma} \cdot 0_{1,0}]_s$. By Corollary 4.9, this function can be extended in a unique way to an idempotent grove theory morphism $\varphi : \mathbf{SRST}_\Sigma \to \mathbf{SRST}_A$ preserving dagger.

It is not hard to see that φ takes the following concrete form. Suppose that $t : 1 \to p$ in $\mathbf{RST}_\Sigma$. Then $[t]_s\varphi$ is the equivalence class of the (regular) tree $t' : 1 \to p$ in $\mathbf{RST}_A$ obtained from t by replacing each edge labeled $\sigma \in \Sigma$ by a copy of the tree s_σ. Formally, the set of vertices of t' consists of the vertices of t together with a vertex $[v, (v_1, v_2, \ldots, v_n)]$ and vertices (v_i, j) with $1 < j < i \leq n$, for each hyper-edge $v \to (v_1, \ldots, v_n)$ of t. The edges of t' are the exit edges of t labeled as in t together with the following ones, where we suppose that $e : v \to (v_1, \ldots, v_n)$ is a hyper-edge of t labeled σ.

(i) An edge $v \to [v, (v_1, \ldots, v_n)]$ labeled $\overline{\sigma}$.

(ii) An edge $[v, (v_1, \ldots, v_n)] \to (v_i, 1)$ for each $1 < i \leq n$ labeled $\#$.

(iii) An edge $(v_i, j) \to (v_i, j+1)$ and an edge $(v_i, i-1) \to v_i$ labeled $\#$, for all $1 < i \leq n$ and $1 < j < i-1$.

(iv) An edge $[v, (v_1, \ldots, v_n)] \to v_1$ labeled $\#$.

When $t = (t_1, \ldots, t_n) : n \to p$ and each $[t_i]_s$ is mapped to $[t'_i]_s$, then $[t]_s\varphi = ([t'_1]_s, \ldots, [t'_n]_s)$. Since each $t : 1 \to p$ can be recovered from $t' : 1 \to p$, $[t]_s$ is uniquely determined by $[t']_s$, i.e., φ is injective. □

Remark 5.2 The above proof can be extended to all synchronization trees to obtain an injective ω-continuous idempotent grove theory morphism $\mathbf{SST}_\Sigma \to \mathbf{SST}_A$.

Proposition 5.3 *For each set A there exist a set B and an injective dagger preserving (idempotent) grove theory morphism* $\mathbf{SRST}_A \to \mathbf{CFL}_B$.

Proof. Let $B = A \cup \{\#, \$\}$ and consider an idempotent grove theory morphism $\varphi : \mathbf{SRST}_A \to \mathbf{CFL}_B$ preserving dagger defined by the assignment $a \mapsto a(\#x_1\$)^* = \{a, a\#x_1\$, a(\#x_1\$)^2, \ldots\} : 1 \to 1$, so that the image of each letter $a \in A$ is a regular language. We claim that for any regular trees $t, s : 1 \to p$ in $\mathbf{RST}_A$,

$$[t] \sqsubseteq_s [s] \Leftrightarrow [t]\varphi \subseteq [s]\varphi.$$

The implication from left-to-right is immediate from Corollary 4.9. Suppose now that $t \not\sqsubseteq_s s$. We want to prove that $[t]\varphi \not\subseteq [s]\varphi$. We consider only the case $p = 0$ since the argument is similar for $p > 0$. The n-round *simulation game* on the pair (t, s) is played by two players, player I and II. In each round, player I selects an edge originating in the vertex of t entered in the previous round, or in the root in the first round, and player II must respond by selecting an equally labeled edge originating in the vertex of s entered in the previous round, or in the root of s in first round. Player I wins the play if player II cannot respond. Otherwise player II wins. Since $t \not\sqsubseteq_s s$, it follows by regularity that there is some $n \geq 1$ such that player I wins the n-round simulation game on (t, s). We show by induction on n that $[t]\varphi \not\subseteq [s]\varphi$. When $n = 1$, player I can choose an edge originating in the root of t whose label is not matched by the label of any edge originating in the root of s. Since the label of this edge is a word in $[t]\varphi$ but not the first letter of any word in $[s]\varphi$, we have $[t]\varphi \not\subseteq [s]\varphi$.

Suppose now that $n > 1$ and that we have established the claim for $n - 1$. Now player I can select an edge originating in the root of t, labeled $a \in A$, with target the root of a subtree t' such that for each a-labeled edge from the root of s to the root of some subtree s', player I wins the $(n-1)$-round game on (t', s'). By the induction hypothesis, this means that $[t']\varphi \not\subseteq [s']\varphi$ for all such subtrees s'.

Let $s_1, \ldots, s_k$ $(k > 0)$be up to isomorphism all those subtrees of s whose roots are the targets of a-labeled edges originating in the root of s. We have $[t']\varphi \not\subseteq [s_i]\varphi$ for all i. Now $[t]\varphi$ contains $a(\#[t']\varphi\$)^k$ as a subset, and all the words in $[s]\varphi$ starting with a are in the set $\{a\} \cup \bigcup_{i \in [k]} \bigcup_{m \geq 1} a(\#[s_i]\varphi\$)^m$. For each $i \in [k]$, let u_i be a word in $[t']\varphi$ which is not in $[s_i]\varphi$. Then the word $a\#u_1\$ \ldots \#u_k\$$ is in $[t]\varphi$ but does not belong to $[s]\varphi$, since it does not belong to any $a(\#[s_i]\varphi)^k$. Thus, $[t]\varphi \not\subseteq [s]\varphi$. □

Proposition 5.4 *For each set A there exist a ranked set Σ and an injective dagger preserving (idempotent) grove theory morphism* $\mathbf{SRST}_A \to \mathbf{Reg}_\Sigma$.

Proof. Let $\Sigma_0 = A \cup \{\#, \$\}$, $\Sigma_2 = \{\sigma\}$, and let Σ_n be empty if $n = 1$ or $n > 2$. For each $a \in A$, consider a regular tree language $L_a : 1 \to 1$ in $\mathbf{Reg}_\Sigma$ whose 'frontier' is the context-free language described in the previous proof. (Such a regular tree language exists, since context-free languages are exactly the frontier languages of regular tree languages, see e.g. [30].) For example, let $L_a = \{t_0 = a, t_1 = \sigma a \sigma \# \sigma x_1 \$, t_2 = \sigma a \sigma \# \sigma x_1 \sigma \$ \sigma \# \sigma x_1 \$, \ldots\}$. Then let $\psi : \mathbf{SRST}_A \to \mathbf{Reg}_\Sigma$ be the unique dagger preserving morphism of idempotent grove theories determined by the assignment $a \mapsto L_a$ for all $a \in A$, which exists by Corollary 4.9. The morphism φ constructed in the proof of Proposition 5.3 factors through ψ by the 'frontier map'. Since φ is injective, so is ψ. □

We are now ready to prove Theorem 3.1. First note that since for each set A, $\mathbf{CFL}_A$ embeds in $\mathbf{Lang}_A$, every identity that holds in all theories $\mathbf{Lang}_A$ holds in the theories $\mathbf{CFL}_A$. Similar facts are true in (iii), (iv) and (v) for the theories $\mathbf{TreeLang}_\Sigma$ and $\mathbf{Reg}_\Sigma$, $\mathbf{SST}_\Sigma$ and $\mathbf{SRST}_\Sigma$, and $\mathbf{SST}_S$ and $\mathbf{SRST}_A$. Clearly, every identity that holds in the theories $\mathbf{SST}_\Sigma$ ($\mathbf{SRST}_\Sigma$, resp.) also holds in the theories $\mathbf{SST}_A$ ($\mathbf{SRST}_A$, resp.). Since each theory $\mathbf{TreeLang}_\Sigma$ embeds in a theory $\mathbf{Lang}_A$, and similarly, each theory $\mathbf{Reg}_\Sigma$ embeds in some theory $\mathbf{CFL}_A$, each condition of (ii) implies the corresponding condition of (iii). By Corollary 4.9, if an identity holds in all theories $\mathbf{SRST}_\Sigma$ then it holds in all ω-continuous idempotent grove theories and thus in all theories appearing in Theorem 3.1. Also, if an identity holds in all power-set theories, then it holds in the theories $\mathbf{Lang}_A$. Thus, to complete the proof it suffices to show that if an identity holds in all theories $\mathbf{Reg}_\Sigma$, then it holds in the theories $\mathbf{SRST}_A$, and that this turn implies that the identity holds in all theories $\mathbf{SRST}_\Sigma$. But these facts follow from Proposition 5.4 and Proposition 5.3.

Remark 5.5 (Based on [19].) The above proof also establishes the fact that an identity holds in all ω-continuous idempotent grove theories iff it holds in all theories $\mathbf{SRST}_\Sigma$ (or the theories mentioned in Theorem 3.1).

When A is a poset with all countable suprema, the ω-continuous functions over A form an ω-continuous idempotent grove theory $\omega\mathbf{Cont}_A$. A morphism $n \to p$ in this theory is an ω-continuous function $A^p \to A^n$ (note the reversal of the arrow), and composition is function composition (in the reverse order). For each $i \in [n]$, $n \geq 0$, the distinguished morphism $i_n : 1 \to n$ is the ith projection function $A^n \to A$. The constant $0_{n,p}$ is the constant function mapping all elements of A^p to the least element of A^n, and $f + g$ is the pointwise supremum of f and g, for each $f, g : n \to p$ (i.e., ω-continuous functions $f, g : A^p \to A^n$). Note that the order $\leq$ becomes the pointwise partial order. Since $\omega\mathbf{Cont}_A$ is a continuous idempotent grove theory, it comes with the least fixed point operation as dagger operation.

Every ω-continuous idempotent grove theory T may be embedded in a theory $\omega\mathbf{Cont}_A$. Given T, let $A = \prod_{p \geq 0} T(1, p)$, equipped with the pointwise partial order. The embedding maps a morphism $f : 1 \to n$ to the ω-continuous function $A^p \to A$ defined by

$$f((g_{1,p})_p, \ldots, (g_{n,p})_p) = (f \cdot \langle g_{1,p}, \ldots, g_{n,p} \rangle)_p.$$

By this embedding, we conclude that an identity holds in all continuous idempotent grove theories iff it holds in all theories of the sort $\omega\mathbf{Cont}_A$, where A is an ω-continuous poset (or in fact a complete lattice).

6 Axiomatization

By Theorem 3.1 and Remark 5.5, the very same equational calculus may be used in formal calculations involving simulation equivalence classes of (regular) synchronization trees, (context-free) languages, (regular) tree languages, (regular) synchronization trees, power-set theories, ω-continuous idempotent grove theories, or ω-continuous functions over ω-complete posets, equipped with the theory operations, sum, and dagger. Axiomatic treatments were given in [19] using the formalism of μ-terms. These results are transformed in [23] into the categorical language of this paper.

Iteration theories were introduced in the late 1970's by Bloom, Elgot and Wright, and independently by Ésik, as a generalization of Elgot's iterative theories [15] and the rational and continuous theories [31,38] of the ADJ group. See [7] for original references. Iteration theories are algebraic theories equipped with a dagger operation satisfying certain equational axioms such as the *fixed point identity*

$$f \cdot \langle f^\dagger, \mathbf{1}_p \rangle = f^\dagger \tag{5}$$

or the *parameter identity*

$$(f \cdot (\mathbf{1}_n \oplus g))^\dagger = f^\dagger \cdot g, \tag{6}$$

where $f : n \to n+p$ and $g : p \to q$. The equational axioms of iteration theories may conveniently be divided into two groups, the 'Conway identities' and the 'commutative identities', which are simplified to the 'group identities' in [18]. For detailed accounts, we refer to [7,9,18]. A morphism of iteration theories is a theory morphism preserving dagger.

A *grove iteration theory* [7,12] is an iteration theory which is a grove theory satisfying

$$\mathbf{1}_1{}^\dagger = 0_{1,0}.$$

It is known that in a grove iteration theory,

$$(\mathbf{1}_n \oplus 0_p)^\dagger = 0_{n,p}$$

holds for all $n, p \geq 0$. Morphisms of grove iteration theories are both iteration theory morphisms and grove theory morphisms.

It is possible to define a *star operation* $^* : T(n, n+p) \to T(n, n+p)$ $(n, p \geq 0)$ in any grove iteration theory. When $f : n \to n+p$, we define

$$f^* = (f \cdot (\mathbf{1}_n \oplus 0_n \oplus \mathbf{1}_p) + (0_n \oplus \mathbf{1}_n \oplus 0_p))^\dagger : n \to n+p.$$

Thus, in particular, $\mathbf{1}_1{}^* = (1_2 + 2_2)^\dagger : 1 \to 1$. It can be seen, cf. [7], that in grove iteration theories T, f^* is a solution of the equation

$$x = f \cdot \langle x, 0_n \oplus \mathbf{1}_p \rangle + (\mathbf{1}_n \oplus 0_p),$$

for all $f : n \to n+p$ in T. When $p = 0$, this equation becomes $x = f \cdot x + \mathbf{1}_n$. In fact, properties of the dagger operation may be translated into corresponding properties of the star operation and vice versa, see [7,22,26].

A grove iteration theory is *ω-idempotent* if

$$\mathbf{1}_1{}^* = \mathbf{1}_1$$

holds. Any ω-idempotent grove iteration theory is an idempotent grove theory and thus an idempotent grove iteration theory. Indeed, if T is ω-idempotent, then

$$\mathbf{1}_1 + \mathbf{1}_1 = \mathbf{1}_1 \cdot \mathbf{1}_1{}^* + \mathbf{1}_1 = \mathbf{1}_1{}^* = \mathbf{1}_1,$$

so that $f + f = (\mathbf{1}_1 + \mathbf{1}_1) \cdot f = \mathbf{1}_1 \cdot f = f$, for all $f : 1 \to p$. Thus, we can define a partial order as above by $f \leq g$ iff $f + g = g$, for all $f, g : n \to p$. Call an idempotent grove iteration theory *ordered* if the dagger operation is monotone:

$$f^\dagger \leq (f + g)^\dagger, \tag{7}$$

or equivalently, if

$$f \leq g : n \to n + p \Rightarrow f^\dagger \leq g^\dagger : n \to p,$$

for all $f, g : n \to n + p$. It follows that composition is also monotone, since in iteration theories,

$$f \cdot g = (\mathbf{1}_n \oplus 0_p) \cdot \langle 0_n \oplus f \oplus 0_q, 0_{n+p} \oplus g \rangle^\dagger,$$

for all $f : n \to p$ and $g : p \to q$. (It can be seen that an idempotent grove theory is ordered iff composition and the scalar dagger operation $f \mapsto f^\dagger$, $f : 1 \to 1 + p$ are monotone.) Morphisms of (ordered) ω-idempotent grove iteration theories are just grove iteration theory morphisms.

The following results were proved in [19] using the formalism of μ-terms. See also [23].

Theorem 6.1 *For each ranked set Σ, $\mathbf{SRST}_\Sigma$ is the free ordered ω-idempotent grove iteration theory, freely generated by Σ.*

Corollary 6.2 *An identity holds in all ordered ω-idempotent grove iteration theories iff it holds in all ω-continuous idempotent grove theories, or in the theories of Theorem 3.1.*

Note that Theorem 6.1 shows that the theories of simulation equivalence classes of regular synchronization theories have a finite axiomatization relatively to iteration theories. (Without the additive structure, they satisfy exactly the iteration theory identities.)

Remark 6.3 By removing (7) from the axioms, we obtain a complete axiomatization of 'bisimilarity' of (regular) synchronization trees, cf. [7,12], and by adding to the the axioms the identities

$$f \cdot (g + h) = f \cdot g + f \cdot h$$
$$f \cdot 0_{p,q} = 0_{n,p},$$

for all $f : n \to p$ and $g, h : p \to q$, the resulting system is known to be complete for (matrix) theories over ω-continuous idempotent semirings, or regular languages, or theories of binary relations, and many other structures. See [7], where original references may be found. (We note that (7) now becomes redundant.)

In the presence of some Conway identities and possibly other axioms, the commutative identities (and the group identities) are implied by simpler axioms. Since many of these holds in ω-continuous idempotent grove theories, we may derive several corollaries to Theorem 6.1 and Corollary 6.2.

A *idempotent Park grove theory* is an ordered idempotent grove theory equipped with a dagger operation satisfying the fixed point identity (5), the parameter identity (6) and the *fixed point induction rule*:

$$f \cdot \langle g, \mathbf{1}_p \rangle \leq g \Rightarrow f^\dagger \leq g,$$

for all $f : n \to n+p$ and $g : n \to p$. For example, all ω-continuous idempotent grove theories are idempotent Park grove theories. It is known, cf. [16], that every idempotent Park grove theory is an ordered ω-idempotent iteration theory. A morphism of idempotent Park grove theories is an idempotent grove theory morphism which preserves dagger.

Using Theorem 6.1 we have:

Corollary 6.4 *For each ranked set Σ, $\mathbf{SRST}_\Sigma$ is the free idempotent Park grove theory, freely generated by Σ.*

Corollary 6.5 *The following are equivalent for an identity $t = t'$ between terms $t, t' : n \to p$ involving the theory operations, dagger, and sum.*

- *$t = t'$ holds in all ω-continuous idempotent grove theories.*
- *$t = t'$ holds in all ordered ω-idempotent grove iteration theories.*
- *$t = t'$ holds in all idempotent Park grove theories.*

It is well-known that it suffices to require the fixed point induction rule just in the case when $n = 1$, see [4,13] or [7].

By the above results, simulation equivalence classes of regular synchronization trees have a finite implicational axiomatization. Other known implicational or first-order axiomatizations involve the weak functorial implication of [7], or a version of the Scott induction rule of [25]. In [22], it is shown that by adding one-sided residuation to the collection of operations, a finite purely equational system may be derived.

References

[1] L. Aceto, Z. Ésik and A. Ingólfsdóttir, Equational axioms for probabilistic bisimilarity, *Algebraic Methodology and Software Technology, AMAST 2002*, Reunion, LNCS 2422, Springer, 2002, 239–253.

[2] J. Balcázar, J. Gabarró, and M. Sántha, Deciding bisimilarity is P-complete, *Formal Aspects of Computing*, 4(1992), 638–648, 1992.

[3] M. Barr and C. Wells, *Category Theory for Computing Science*, Prentice Hall, 1990.

[4] H. Bekić, Definable operation in general algebras, and the theory of automata and flowcharts, Technical report, IBM Vienna, 1969. Reprinted in: *Programming Languages and Their Definition – Hans Bekic (1936-1982)*, LNCS 177, Springer 1984, 30–55.

[5] S.L. Bloom, Varieties of ordered algebras, *J. Computer and System Sciences*, 13(1976), 200–212.

[6] S.L. Bloom and Z.Ésik, Equational logic of circular data type specification, *Theoret. Comput. Sci.*, 63(1989), 303–331.

[7] S.L. Bloom and Z. Ésik, *Iteration Theories*, Springer, 1993.

[8] S.L. Bloom and Z. Ésik, Equational axioms for regular sets, *Mathematical Structures in Computer Science*, 3(1993), 1–24.

[9] S.L. Bloom and Z. Ésik, The equational logic of fixed points, *Theoretical Computer Science*, 179(1997), 1–60.

[10] S.L. Bloom and Z. Ésik, An extension theorem with an application to formal tree series, *J. Autmata, Languages, and Combinatorics*, 8(2003), 145–185.

[11] S.L. Bloom and Z. Ésik, Axiomatizing rational power series over natural numbers, *Information and Computation*, 207(2009), 793–811.

[12] S.L. Bloom, Z. Ésik and D. Taubner, Iteration theories of synchronization trees, *Inform. and Comput.*, 102(1993), 1–55.

[13] J.W. DeBakker and D. Scott, A theory of programs, *Technical Report, IBM Vienna*, 1969.

[14] J. Engelfriet and E.M. Schmidt, IO and OI, Parts I and II, *J. Comput. System Sci.*, 15(1977), 328–353 and 16(1978), 67–99.

[15] C.C. Elgot, Monadic computation and iterative algebraic theories, *Logic Colloquium '73 (Bristol, 1973)*, Studies in Logic and the Foundations of Mathematics, Vol. 80, North-Holland, Amsterdam, 1975, 175–230.

[16] Z. Ésik, Completeness of Park induction, *Theoret. Comput. Sci.*, 177(1997), 217–283.

[17] Z. Ésik, Axiomatizing the equational theory of regular tree languages, *STACS '98, Paris,* LNCS 1373, Springer-Verlag, 1998, 455-465.

[18] Z. Ésik, Group axioms for iteration, *Inform. and Comput.*, 148(1999), 131–180.

[19] Z. Ésik, Axiomatizing the least fixed point operation and binary supremum, *Computer Science Logic (Fischbachau, 2000)*, LNCS 1862, 2000, 302–316.

[20] Z. Ésik, Continuous additive algebras and injective simulations of synchronization trees, *J. Logic. Comput.*, 12(2002), 271–300.

[21] Z. Ésik, Axiomatizing the equational theory of regular tree languages, *J. Logic and Algebraic Programming*, 79(2010), 189–213.

[22] Z. Ésik, Residuated Park theories, *J. Logic Computation*, published on line on Feb. 5, 2013.

[23] Z. Ésik, A connection between concurrency and language theory, arXiv:1303.0044, March 1, 2013.

[24] Z. Ésik, Axiomatizing weighted bisimulation, *Theoretical Computer Science*, to appear.

[25] Z. Ésik and L. Bernátsky, Scott induction and equational proofs, *Mathematical Foundations of Programming Semantics*, New Orleans, ENTCS 1(1995), 32 pages, available at: http://www.elsevier.nl.

[26] Z. Ésik and T. Hajgató, Iteration grove theories with applications, *CAI 2009, Thessaloniki*, LNCS 5725, Springer, 2009, 227–249.

[27] Z. Ésik and W. Kuich, Free iterative and iteration K-semialgebras, *Algebra Universalis*, 67(2012), 141–162.

[28] Z. Ésik and W. Kuich, Free inductive K-semialgebras, *J. Logic and Algebraic Programming*, published online on 18 January 2013.

[29] Z. Ésik and A. Labella, Equational properties of iteration in algebraically complete categories, *Theoret. Comput. Sci.*, 195(1998), 61–89.

[30] F. Gécseg and M. Steinby, *Tree Automata*, Akadémiai Kiadó, Budapest, 1986.

[31] J.A. Goguen, J.W. Thatcher, E.G. Wagner and J.B. Wright, Initial algebra semantics and continuous algebras, *J. ACM*, 24(1977), 68–95.

[32] W.F. Lawvere, Functorial semantics of algebraic theories, *Proc. Nat. Acad. Sci. U.S.A.*, 50(1963) 869–872.

[33] R. Milner, *Communication and Concurrency*, Prentice Hall, 1989.

[34] D. Park, Concurrency and automata on infinite sequences, in: *Proc. 5th GI-Conference, Karlsruhe*, LNCS 104, Springer, 1981, 167–183.

[35] A.K. Simpson and G.D. Plotkin, Complete axioms for categorical fixed-point operators, *LICS 2000*, IEEE Press, 2000, 30–41.

[36] Z. Sawa and P. Jancar, P-hardness of equivalence testing on finite-state processes, *SOFSEM 2001*, LNCS 2234, Springer, 2001, 326–335.

[37] G. Winskel, Synchronization trees, *Theoret. Comput. Sci.*, 34(1984), 33–82.

[38] J.B. Wright, J.W. Thatcher, E.G. Wagner and J.A. Goguen, Rational algebraic theories and fixed-point solutions, *17th Ann. Symp. Foundations of Computer Science, FOCS 17*, IEEE Press, 1976, 147–158.

History-Preserving Bisimilarity for Higher-Dimensional Automata via Open Maps

Uli Fahrenberg and Axel Legay

INRIA/IRISA, Campus de Beaulieu, 35042 Rennes CEDEX, France

Abstract

We show that history-preserving bisimilarity for higher-dimensional automata has a simple characterization directly in terms of higher-dimensional transitions. This implies that it is decidable for finite higher-dimensional automata. To arrive at our characterization, we apply the open-maps framework of Joyal, Nielsen and Winskel in the category of unfoldings of precubical sets.

Keywords: higher-dimensional automaton, history-preserving bisimilarity, homotopy, unfolding, concurrency

1 Introduction

The dominant notion for behavioral equivalence of processes is *bisimulation* as introduced by Park [31] and Milner [27]. It is compelling because it enjoys good algebraic properties, admits several easy characterizations using modal logics, fixed points, or game theory, and generally has low computational complexity.

Bisimulation, or rather its underlying semantic model of *transition systems*, applies to a setting in which concurrency of actions is the same as non-deterministic interleaving; using CCS notation [27], $a|b = a.b+b.a$. For some applications however, a distinction between these two is necessary, which has led to development of so-called *non-interleaving* or *truly concurrent* models such as Petri nets [32], event structures [30], asynchronous transition systems [4, 35] and others; see [40] for a survey.

One of the most popular notions of equivalence for non-interleaving systems is *history-preserving bisimilarity* (or *hp-bisimilarity* for short). It was introduced independently by Degano, De Nicola and Montanari in [6] and by Rabinovich and Trakhtenbrot [34] and then for event structures by van Glabbeek and Goltz in [39] and for Petri nets by Best *et.al.* in [5]. One reason for its popularity is that it is a congruence under action refinement [5, 39], another its good decidability properties: it has been shown to be decidable for safe Petri nets by Montanari and Pistore [29].

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: `www.elsevier.com/locate/entcs`

As a contrast, its cousin *hereditary* hp-bisimilarity is shown undecidable for 1-safe Petri nets by Jurdziński, Nielsen and Srba in [24].

Higher-dimensional automata (or *HDA*) is another non-interleaving formalism for reasoning about behavior of concurrent systems. Introduced by Pratt [33] and van Glabbeek [37] in 1991 for the purpose of a *geometric* interpretation to the theory of concurrency, it has since been shown by van Glabbeek [38] that HDA provide a generalization (up to hp-bisimilarity) to "the main models of concurrency proposed in the literature" [38], including the ones mentioned above. Hence HDA are useful as a tool for comparing and relating different models, and also as a modeling formalism by themselves.

HDA are geometric in the sense that they are very similar to the *simplicial complexes* used in algebraic topology, and research on HDA has drawn on a lot of tools and methods from geometry and algebraic topology such as homotopy [11,14], homology [15,20], and model categories [16,17], see also the survey [18].

In this paper we give a geometric interpretation to hp-bisimilarity for HDA, using the open-maps approach introduced by Joyal, Nielsen and Winskel in [23] and results from a previous paper [7] by the first author. Using this interpretation, we show that hp-bisimilarity for HDA has a characterization directly in terms of (higher-dimensional) *transitions* of the HDA, rather than in terms of runs as *e.g.* for Petri nets [13].

Our results imply *decidability* of hp-bisimilarity for finite HDA. They also put hp-bisimilarity firmly into the open-maps framework of [23] and tighten the connections between bisimilarity and weak topological *fibrations* [3,25].

Due to lack of space, we have had to omit all proofs in this paper. They can be found in the long version at [9].

2 Higher-Dimensional Automata

As a formalism for concurrent behavior, HDA have the specific feature that they can express all higher-order dependencies between events in a concurrent system. Like for transition systems, they consist of states and transitions which are labeled with events. Now if two transitions from a state, with labels a and b for example, are independent, then this is expressed by the existence of a *two-dimensional* transition with label ab. Fig. 1 shows two examples; on the left, transitions a and b are independent, on the right, they can merely be executed in any order. Hence for HDA, as indeed for any formalism employing the so-called *true concurrency* paradigm, the algebraic law $a|b = a.b + b.a$ does *not* hold; concurrency is not the same as interleaving.

The above considerations can equally be applied to sets of more than two events: if three events a, b, c are independent, then this is expressed using a three-dimensional transition labeled abc. Hence this is different from mutual pairwise independence (expressed by transitions ab, ac, bc), a distinction which cannot be made in formalisms such as asynchronous transition systems [4,35] or transition systems with independence [40] which only consider binary independence relations.

An unlabeled HDA is essentially a pointed precubical set as defined below. For labeled HDA, one can pass to an arrow category; this is what we shall do in Section 6.

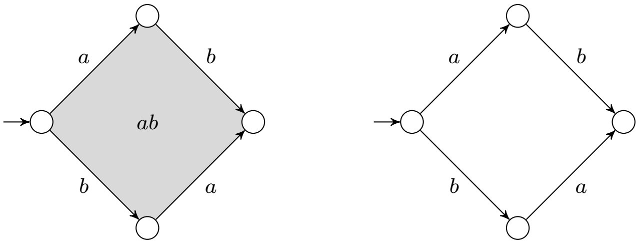


Fig. 1. HDA for the CCS expressions $a|b$ (left) and $a.b+b.a$ (right). In the left HDA, the square is filled in by a two-dimensional transition labeled ab, signifying independence of events a and b. On the right, a and b are not independent.

Until then, we concentrate on the unlabeled case.

A *precubical set* is a graded set $X = \{X_n\}_{n\in\mathbb{N}}$ together with mappings $\delta_k^\nu : X_n \to X_{n-1}$, $k \in \{1, \dots, n\}$, $\nu \in \{0,1\}$, satisfying the *precubical identity*

$$\delta_k^\nu \delta_\ell^\mu = \delta_{\ell-1}^\mu \delta_k^\nu \qquad (k < \ell)\,. \tag{1}$$

The mappings δ_k^ν are called *face maps*, and elements of X_n are called *n-cubes*. As above, we shall usually omit the extra subscript (n) in the face maps. Faces $\delta_k^0 x$ of an element $x \in X$ are to be thought of as *lower faces*, $\delta_k^1 x$ as *upper faces*. The precubical identity expresses the fact that $(n-1)$-faces of an n-cube meet in common $(n-2)$-faces, see Fig. 2 for an example of a 2-cube and its faces.

Morphisms $f : X \to Y$ of precubical sets are graded mappings $f = \{f_n : X_n \to Y_n\}_{n\in\mathbb{N}}$ which commute with the face maps: $\delta_k^\nu \circ f_n = f_{n-1} \circ \delta_k^\nu$ for all $n \in \mathbb{N}$, $k \in \{1, \dots, n\}$, $\nu \in \{0,1\}$. This defines a category pCub of precubical sets and morphisms.

A *pointed* precubical set is a precubical set X with a specified 0-cube $i \in X_0$, and a pointed morphism is one which respects the point. This defines a category which is isomorphic to the comma category $* \downarrow \mathsf{pCub}$, where $* \in \mathsf{pCub}$ is the precubical set with one 0-cube and no other n-cubes. Note that $*$ is *not* terminal in pCub (instead, the terminal object is the infinite-dimensional precubical set with one cube in every dimension).

Definition 2.1 The category of *higher-dimensional automata* is the comma category $\mathsf{HDA} = * \downarrow \mathsf{pCub}$, with objects pointed precubical sets and morphisms commutative diagrams

$$\begin{tikzcd}
 & * \arrow[dl] \arrow[dr] & \\
X \arrow[rr, "f"'] & & Y\,.
\end{tikzcd}$$

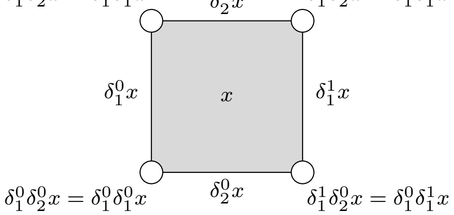

$\delta_1^0\delta_2^1 x = \delta_1^1\delta_1^0 x$
$\delta_2^1 x$
$\delta_1^1\delta_2^1 x = \delta_1^1\delta_1^1 x$
$\delta_1^0 x$
x
$\delta_1^1 x$
$\delta_1^0\delta_2^0 x = \delta_1^0\delta_1^0 x$
$\delta_2^0 x$
$\delta_1^1\delta_2^0 x = \delta_1^0\delta_1^1 x$

Fig. 2. A 2-cube x with its four faces $\delta_1^0 x$, $\delta_1^1 x$, $\delta_2^0 x$, $\delta_2^1 x$ and four corners.

Hence a one-dimensional HDA is a transition system; indeed, the category of transition systems [40] is isomorphic to the full subcategory of HDA spanned by the one-dimensional objects. Similarly one can show [19] that the category of asynchronous transition systems is isomorphic to the full subcategory of HDA spanned by the (at most) two-dimensional objects. The category HDA as defined above was used in [7] to provide a categorical framework (in the spirit of [40]) for parallel composition of HDA. In this article we also introduced a notion of bisimilarity which we will review in the next section.

3 Path Objects, Open Maps and Bisimilarity

With the purpose of introducing bisimilarity via *open maps* in the sense of [23], we identify here a subcategory of HDA consisting of path objects and path-extending morphisms. We say that a precubical set X is a *precubical path object* if there is a (necessarily unique) sequence $(x_1, \dots, x_m)$ of elements in X such that $x_i \neq x_j$ for $i \neq j$,

- for each $x \in X$ there is $j \in \{1, \dots, m\}$ for which $x = \delta_{k_1}^{\nu_1} \cdots \delta_{k_p}^{\nu_p} x_j$ for some indices $\nu_1, \dots, \nu_p$ and a *unique* sequence $k_1 < \cdots < k_p$, and
- for each $j = 1, \dots, m-1$, there is $k \in \mathbb{N}$ for which $x_j = \delta_k^0 x_{j+1}$ or $x_{j+1} = \delta_k^1 x_j$.

Note that precubical path objects are *non-selflinked* in the sense of [11]. If X and Y are precubical path objects with representations $(x_1, \dots, x_m)$, $(y_1, \dots, y_p)$, then a morphism $f : X \to Y$ is called a *cube path extension* if $x_j = y_j$ for all $j = 1, \dots, m$ (hence $m \leq p$).

Definition 3.1 The category HDP of *higher-dimensional paths* is the subcategory of HDA which as objects has pointed precubical paths, and whose morphisms are generated by isomorphisms and pointed cube path extensions.

A *cube path* in a precubical set X is a morphism $P \to X$ from a precubical path object P. In elementary terms, this is a sequence $(x_1, \dots, x_m)$ of elements of X such that for each $j = 1, \dots, m-1$, there is $k \in \mathbb{N}$ for which $x_j = \delta_k^0 x_{j+1}$ (start of new part of a computation) or $x_{j+1} = \delta_k^1 x_j$ (end of a computation part). We show an example of a cube path in Fig. 3.

A cube path in a HDA $i : * \to X$ is *pointed* if $x_1 = i$, hence if it is a pointed morphism $P \to X$ from a higher-dimensional path P. We will say that a cube path $(x_1, \dots, x_m)$ is *from x_1 to x_m*, and that a cube $x \in X$ in a HDA X is *reachable* if there is a pointed cube path to x in X.

Cube paths can be *concatenated* if the end of one is compatible with the beginning of the other: If $\rho = (x_1, \dots, x_m)$ and $\sigma = (y_1, \dots, y_p)$ are cube paths with $y_1 = \delta_k^1 x_m$ or $x_m = \delta_k^0 y_1$ for some k, then their *concatenation* is the cube path $\rho * \sigma = (x_1, \dots, x_m, y_1, \dots, y_p)$. We say that ρ is a *prefix* of χ and write $\rho \sqsubseteq \chi$ if there is a cube path ρ for which $\chi = \rho * \sigma$.

Definition 3.2 A pointed morphism $f : X \to Y$ in HDA is an *open map* if it has the right lifting property with respect to HDP, *i.e.* if it is the case that there is a lift r in any commutative diagram as below, for morphisms $g : P \to Q \in$ HDP,

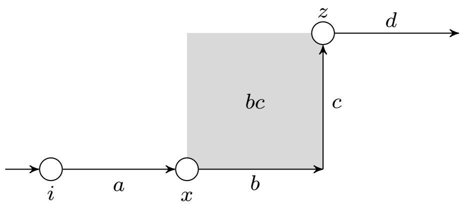


Fig. 3. Graphical representation of the two-dimensional cube path $(i, a, x, b, bc, c, z, d)$. Its computational interpretation is that a is executed first, then execution of b starts, and while b is running, c starts to execute. After this, b finishes, then c, and then execution of d is started. Note that the computation is partial, as d does not finish.

$p : P \to X, q : Q \to Y \in \mathsf{HDA}$:

$$\begin{tikzcd} P \arrow[r, "p"] \arrow[d, "g"'] & X \arrow[d, "f"] \\ Q \arrow[r, "q"'] \arrow[ur, dotted, "r"] & Y \end{tikzcd}$$

HDA X, Y are *bisimilar* if there is $Z \in \mathsf{HDA}$ and a span of open maps $X \leftarrow Z \to Y$ in HDA.

It follows straight from the definition that composites of open maps are again open. By the next lemma, morphisms are open precisely when they have a zig-zag property similar to the one of [23].

Lemma 3.3 *For a morphism $f : X \to Y \in \mathsf{HDA}$, the following are equivalent:*

(i) *f is open;*

(ii) *for any reachable $x_1 \in X$ and any $y_2 \in Y$ with $f(x_1) = \delta^0_k y_2$ for some k, there is $x_2 \in X$ for which $x_1 = \delta^0_k x_2$ and $y_2 = f(x_2)$;*

(iii) *for any reachable $x_1 \in X$ and any cube path $(y_1, \dots, y_m)$ in Y with $y_1 = f(x_1)$, there is a cube path $(x_1, \dots, x_m)$ in X for which $y_j = f(x_j)$ for all $j = 1, \dots, m$.*

Theorem 3.4 *For HDA $i : * \to X$, $j : * \to Y$, the following are equivalent:*

(i) *X and Y are bisimilar;*

(ii) *there exists a precubical subset $R \subseteq X \times Y$ for which $(i, j) \in R$, and such that for all reachable $x_1 \in X$, $y_1 \in Y$ with $(x_1, y_1) \in R$,*
- *for any $x_2 \in X$ for which $x_1 = \delta^0_k x_2$ for some k, there exists $y_2 \in Y$ for which $y_1 = \delta^0_k y_2$ and $(x_2, y_2) \in R$,*
- *for any $y_2 \in Y$ for which $y_1 = \delta^0_k y_2$ for some k, there exists $x_2 \in X$ for which $x_1 = \delta^0_k x_2$ and $(x_2, y_2) \in R$;*

(iii) *there exists a precubical subset $R \subseteq X \times Y$ for which $(i, j) \in R$, and such that for all reachable $x_1 \in X$, $y_1 \in Y$ with $(x_1, y_1) \in R$,*
- *for any cube path $(x_1, \dots, x_m)$ in X, there exists a cube path $(y_1, \dots, y_m)$ in Y with $(x_p, y_p) \in R$ for all $p = 1, \dots, m$,*
- *for any cube path $(y_1, \dots, y_m)$ in Y, there exists a cube path $(x_1, \dots, x_m)$ in X with $(x_p, y_p) \in R$ for all $p = 1, \dots, m$.*

Note that the requirement that R be a precubical subset, in items (ii) and (iii) above, is equivalent to saying that whenever $(x, y) \in R$, then also $(\delta^\nu_k x, \delta^\nu_k y) \in R$ for any k and $\nu \in \{0, 1\}$.

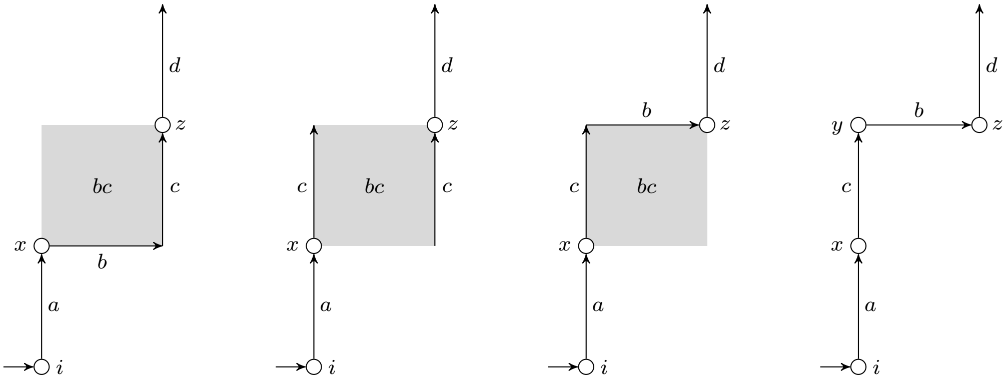


Fig. 4. Graphical representation of the cube path homotopy $(i, a, x, b, bc, c, z, d) \sim (i, a, x, c, bc, c, z, d) \sim (i, a, x, c, bc, b, z, d) \sim (i, a, x, c, y, b, z, d)$.

4 Homotopies and Unfoldings

In order to reason about hp-bisimilarity, we need to introduce in which cases different cube paths are equivalent due to independence of actions. Following [38], we model this equivalence by a combinatorial version of *homotopy* which is an extension of the equivalence defining *Mazurkiewicz traces* [26].

We say that cube paths $(x_1, \dots, x_m)$, $(y_1, \dots, y_m)$ are *adjacent* if $x_1 = y_1$, $x_m = y_m$, there is precisely one index $p \in \{1, \dots, m\}$ at which $x_p \neq y_p$, and

- $x_{p-1} = \delta^0_k x_p$, $x_p = \delta^0_\ell x_{p+1}$, $y_{p-1} = \delta^0_{\ell-1} y_p$, and $y_p = \delta^0_k y_{p+1}$ for some $k < \ell$, or vice versa,
- $x_p = \delta^1_k x_{p-1}$, $x_{p+1} = \delta^1_\ell x_p$, $y_p = \delta^1_{\ell-1} y_{p-1}$, and $y_{p+1} = \delta^1_k y_p$ for some $k < \ell$, or vice versa,
- $x_p = \delta^0_k \delta^1_\ell y_p$, $y_{p-1} = \delta^0_k y_p$, and $y_{p+1} = \delta^1_\ell y_p$ for some $k < \ell$, or vice versa, or
- $x_p = \delta^1_k \delta^0_\ell y_p$, $y_{p-1} = \delta^0_\ell y_p$, and $y_{p+1} = \delta^1_k y_p$ for some $k < \ell$, or vice versa.

Homotopy of cube paths is the reflexive, transitive closure of the adjacency relation. We denote homotopy of cube paths using the symbol $\sim$, and the homotopy class of a cube path $(x_1, \dots, x_m)$ is denoted $[x_1, \dots, x_m]$. The intuition of adjacency is rather simple, even though the combinatorics may look complicated, see Fig. 4. Note that adjacencies come in two basic "flavors": the first two above in which the dimensions of x_ℓ and y_ℓ are the same, and the last two in which they differ by 2.

The following lemma shows that, as expected, cube paths entirely contained in one cube are homotopic (provided that they share endpoints).

Lemma 4.1 *Let $x \in X_n$ in a precubical set X and $(k_1, \dots, k_n)$, $(\ell_1, \dots, \ell_n)$ sequences of indices with $k_j, \ell_j \le j$ for all $j = 1, \dots, n$. Let $x_j = \delta^0_{k_j} \cdots \delta^0_{k_n} x$, $y_j = \delta^0_{\ell_j} \cdots \delta^0_{\ell_n} x$. Then the cube paths $(x_1, \dots, x_n, x) \sim (y_1, \dots, y_n, x)$.*

We extend concatenation and prefix to homotopy classes of cube paths by defining $[x_1, \dots, x_m] * [y_1, \dots, y_p] = [x_1, \dots, x_m, y_1, \dots, y_p]$ and saying that $\tilde x \sqsubseteq \tilde z$, for homotopy classes $\tilde x$, $\tilde z$ of cube paths, if there are $(x_1, \dots, x_m) \in \tilde x$ and $(z_1, \dots, z_q) \in \tilde z$ for which $(x_1, \dots, x_m) \sqsubseteq (z_1, \dots, z_q)$. It is easy to see that concatenation is well-defined, and that $\tilde x \sqsubseteq \tilde z$ if and only if there is a homotopy class $\tilde y$ for which $\tilde z = \tilde x * \tilde y$.

Using homotopy classes of cube paths, we can now define the *unfolding* of a

HDA. Unfoldings of HDA are similar to unfoldings of transition systems [40] or Petri nets [22,30], but also to *universal covering spaces* in algebraic topology. The intention is that the unfolding of a HDA captures all its computations, up to homotopy.

We say that a HDA X is a *higher-dimensional tree* if it holds that for any $x \in X$, there is precisely one homotopy class of pointed cube paths to x. The full subcategory of HDA spanned by the higher-dimensional trees is denoted HDT. Note that any higher-dimensional path is a higher-dimensional tree; indeed there is an inclusion $\mathsf{HDP} \hookrightarrow \mathsf{HDT}$.

Definition 4.2 The *unfolding* of a HDA $i : * \to X$ consists of a HDA $\tilde{i} : * \to \tilde{X}$ and a pointed *projection* morphism $\pi_X : \tilde{X} \to X$, which are defined as follows:

- $\tilde{X}_n = \big\{[x_1, \dots, x_m] \mid (x_1, \dots, x_m)$ pointed cube path in $X, x_m \in X_n\big\}$; $\tilde{i} = [i]$
- $\tilde{\delta}^0_k[x_1, \dots, x_m] = \big\{\sigma = (y_1, \dots, y_p) \mid y_p = \delta^0_k x_m, \sigma * x_m \sim (x_1, \dots, x_m)\big\}$
- $\tilde{\delta}^1_k[x_1, \dots, x_m] = [x_1, \dots, x_m, \delta^1_k x_m]$
- $\pi_X[x_1, \dots, x_m] = x_m$

Proposition 4.3 *The unfolding $(\tilde{X}, \pi_X)$ of a HDA X is well-defined, and $\tilde{X}$ is a higher-dimensional tree. If X itself is a higher-dimensional tree, then the projection $\pi_X : \tilde{X} \to X$ is an isomorphism.*

Lemma 4.4 *If X is a higher-dimensional automaton and $(\tilde{x}_1, \dots, \tilde{x}_m)$ is a pointed cube path in $\tilde{X}$, then $(\pi_X \tilde{x}_1, \dots, \pi_X \tilde{x}_j) \in \tilde{x}_j$ for all $j = 1, \dots, m$.*

Lemma 4.5 *For any HDA X there is a unique lift r in any commutative diagram as below, for morphisms $g : P \to Q \in \mathsf{HDP}$, $p : P \to \tilde{X}, q : Q \to X \in \mathsf{HDA}$:*

$$\begin{tikzcd}
P \arrow[r, "p"] \arrow[d, "g"'] & \tilde{X} \arrow[d, "\pi_X"] \\
Q \arrow[r, "q"'] \arrow[ur, dotted, "r"] & X
\end{tikzcd}$$

Corollary 4.6 *Projections are open, and any HDA is bisimilar to its unfolding.* □

5 History-Preserving Bisimilarity

In this section we recall history-preserving bisimilarity for HDA from [38] and show the main result of this paper: that hp-bisimilarity and the bisimilarity of Def. 3.2 are the same. To do this, we first need to introduce *morphisms of homotopy classes of paths* and *homotopy bisimilarity*.

Definition 5.1 The category of *higher-dimensional automata up to homotopy* $\mathsf{HDA_h}$ has as objects HDA and as morphisms pointed precubical morphisms $f : \tilde{X} \to \tilde{Y}$ of unfoldings.

Hence any morphism $X \to Y$ in HDA gives, by the unfolding functor, rise to a morphism $X \to Y$ in $\mathsf{HDA_h}$. The simple example in Fig. 5 shows that the converse is not the case. By restriction to higher-dimensional trees, we get a full subcategory $\mathsf{HDT_h} \hookrightarrow \mathsf{HDA_h}$.

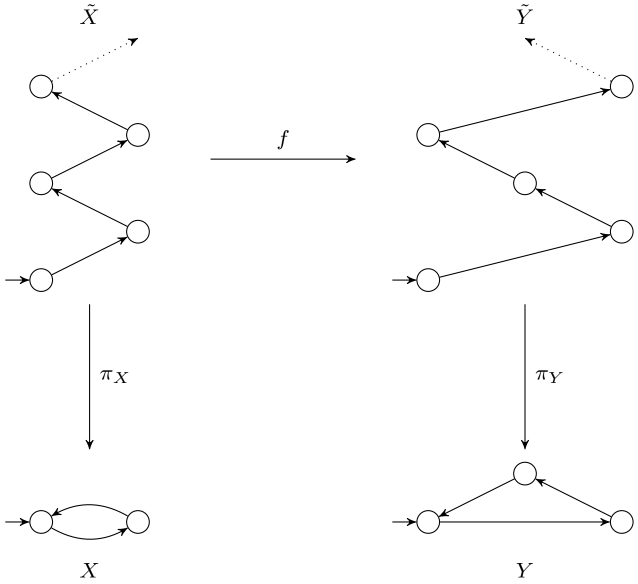


Fig. 5. Two simple one-dimensional HDA as objects of HDA and $\mathsf{HDA_h}$. In HDA there is no morphism $X \to Y$, in $\mathsf{HDA_h}$ there is precisely one morphism $f : X \to Y$.

Lemma 5.2 *The natural projection isomorphisms* $\pi_X : \tilde{X} \to X$ *for* $X \in \mathsf{HDT}$ *extend to an isomorphism of categories* $\mathsf{HDT_h} \cong \mathsf{HDT}$.

Restricting the above isomorphism to the subcategory HDP of HDT allows us to identify a subcategory $\mathsf{HDP_h}$ of $\mathsf{HDT_h}$ isomorphic to HDP.

Definition 5.3 A pointed morphism $f : X \to Y$ in $\mathsf{HDA_h}$ is *open* if it has the right lifting property with respect to $\mathsf{HDP_h}$, *i.e.* if it is the case that there is a lift r in any commutative diagram as below, for all morphism $g : P \to Q \in \mathsf{HDP_h}$, $p : P \to X, q : Q \to Y \in \mathsf{HDA_h}$:

$$\begin{tikzcd}
P \arrow[r, "p"] \arrow[d, "g"'] & X \arrow[d, "f"] \\
Q \arrow[r, "q"'] \arrow[ur, dotted, "r"] & Y
\end{tikzcd}$$

HDA X, Y are *homotopy bisimilar* if there is $Z \in \mathsf{HDA_h}$ and a span of open maps $X \leftarrow Z \to Y$ in $\mathsf{HDA_h}$.

The connections between open maps in $\mathsf{HDA_h}$ and open maps in HDA are as follows.

Lemma 5.4 *A morphism* $f : X \to Y$ *in* $\mathsf{HDA_h}$ *is open if and only if* $f : \tilde{X} \to \tilde{Y}$ *is open as a morphism of* HDA. *If* $g : X \to Y$ *is open in* HDA, *then so is* $\tilde{g} : \tilde{X} \to \tilde{Y}$.

We also need a lemma on prefixes in unfoldings.

Lemma 5.5 *Let* X *be a HDA and* $\tilde{x}, \tilde{z} \in \tilde{X}$. *Then there is a cube path from* $\tilde{x}$ *to* $\tilde{z}$ *in* $\tilde{X}$ *if and only if* $\tilde{x} \sqsubseteq \tilde{z}$.

Proposition 5.6 *For HDA* $i : * \to X$, $j : * \to Y$, *the following are equivalent:*

(i) *X and Y are homotopy bisimilar;*

(ii) *there exists a precubical subset $R \subseteq \tilde X \times \tilde Y$ with $(\tilde\imath, \tilde\jmath) \in R$, and such that for all $(\tilde x_1, \tilde y_1) \in R$,*
- *for any $\tilde x_2 \in \tilde X$ for which $\tilde x_1 = \delta_k^0 \tilde x_2$ for some k, there exists $\tilde y_2 \in \tilde Y$ for which $\tilde y_1 = \delta_k^0 \tilde y_2$ and $(\tilde x_2, \tilde y_2) \in R$,*
- *for any $\tilde y_2 \in \tilde Y$ for which $\tilde y_1 = \delta_k^0 \tilde y_2$ for some k, there exists $\tilde x_2 \in \tilde X$ for which $\tilde x_1 = \delta_k^0 \tilde x_2$ and $(\tilde x_2, \tilde y_2) \in R$;*

(iii) *there exists a precubical subset $R \subseteq \tilde X \times \tilde Y$ with $(\tilde\imath, \tilde\jmath) \in R$, and such that for all $(\tilde x_1, \tilde y_1) \in R$,*
- *for any cube path $(\tilde x_1, \dots, \tilde x_n)$ in $\tilde X$, there exists a cube path $(\tilde y_1, \dots, \tilde y_n)$ in $\tilde Y$ with $(\tilde x_p, \tilde y_p) \in R$ for all $p = 1, \dots, n$,*
- *for any cube path $(\tilde y_1, \dots, \tilde y_n)$ in $\tilde Y$, there exists a cube path $(\tilde x_1, \dots, \tilde x_n)$ in $\tilde X$ with $(\tilde x_p, \tilde y_p) \in R$ for all $p = 1, \dots, n$;*

(iv) *there exists a precubical subset $R \subseteq \tilde X \times \tilde Y$ with $(\tilde\imath, \tilde\jmath) \in R$, and such that for all $(\tilde x_1, \tilde y_1) \in R$,*
- *for any $\tilde x_2 \sqsupseteq \tilde x_1$ in $\tilde X$, there exists $\tilde y_2 \sqsupseteq \tilde y_1$ in $\tilde Y$ for which $(\tilde x_2, \tilde y_2) \in R$,*
- *for any $\tilde y_2 \sqsupseteq \tilde y_1$ in $\tilde Y$, there exists $\tilde x_2 \sqsupseteq \tilde x_1$ in $\tilde X$ for which $(\tilde x_2, \tilde y_2) \in R$.*

Again, the requirement that R be a precubical subset is equivalent to saying that whenever $(\tilde x, \tilde y) \in R$, then also $(\delta_k^\nu \tilde x, \delta_k^\nu \tilde y) \in R$ for any k and $\nu \in \{0, 1\}$. The next result is what will allow us to relate hp-bisimilarity and bisimilarity.

Theorem 5.7 *HDA X, Y are homotopy bisimilar if and only if they are bisimilar.*

The following is an unlabeled version of hp-bisimilarity for HDA as defined in [38]:

Definition 5.8 HDA $i : * \to X$, $j : * \to Y$ are *history-preserving bisimilar* if there exists a relation R between pointed cube paths in X and pointed cube paths in Y for which $((i), (j)) \in R$, and such that for all $(\rho, \sigma) \in R$,

- for all $\rho' \sim \rho$, there exists $\sigma' \sim \sigma$ with $(\rho', \sigma') \in R$,
- for all $\sigma' \sim \sigma$, there exists $\rho' \sim \rho$ with $(\rho', \sigma') \in R$,
- for all $\rho' \sqsupseteq \rho$, there exists $\sigma' \sqsupseteq \sigma$ with $(\rho', \sigma') \in R$,
- for all $\sigma' \sqsupseteq \sigma$, there exists $\rho' \sqsupseteq \rho$ with $(\rho', \sigma') \in R$.

We are ready to show the main result of this paper, which together with Theorem 5.7 gives our characterization for hp-bisimilarity.

Theorem 5.9 *HDA X, Y are homotopy bisimilar if and only if they are history-preserving bisimilar.*

Corollary 5.10 *History-preserving bisimilarity is decidable for finite HDA.*

6 Labels

We finish this paper by showing how to introduce labels into the above framework of bisimilarity and homotopy bisimilarity. Also in the labeled case, we are able to show that the three notions of bisimilarity, homotopy bisimilarity and history-preserving bisimilarity agree.

For labeling HDA, we need a subcategory of pCub isomorphic to the category of sets and functions. Given a finite or countably infinite set $S = \{a_1, a_2, \dots\}$, we construct a precubical set $!S = \{!S_n\}$ by letting

$$!S_n = \big\{(a_{i_1}, \dots, a_{i_n}) \mid i_k \le i_{k+1} \text{ for all } k = 1, \dots, n-1\big\}$$

with face maps defined by $\delta_k^\nu(a_{i_1}, \dots, a_{i_n}) = (a_{i_1}, \dots, a_{i_{k-1}}, a_{i_{k+1}}, \dots, a_{i_n})$.

Definition 6.1 The category of *higher-dimensional tori* HDO is the full subcategory of pCub generated by the objects $!S$.

As any object in HDO has precisely one 0-cube, the pointed category $* \downarrow \mathsf{HDO}$ is isomorphic to HDO. It is not difficult to see that HDO is indeed isomorphic to the category of finite or countably infinite sets and functions, *cf.* [21].

Definition 6.2 The category of *labeled higher-dimensional automata* is the pointed arrow category $\mathsf{LHDA} = * \downarrow \mathsf{pCub} \to \mathsf{HDO}$, with objects $* \to X \to\ !S$ labeled pointed precubical sets and morphisms commutative diagrams

$$\begin{tikzcd}
 & * \arrow[dl] \arrow[dr] & \\
X \arrow[rr, "f"'] \arrow[d] & & Y \arrow[d] \\
!S \arrow[rr, "\sigma"'] & & !T
\end{tikzcd}$$

Definition 6.3 A morphism $(f, \mathrm{id}) : (* \to X \to\ !S) \to (* \to Y \to\ !S)$ in LHDA is *open* if its component f is open in HDA. Labeled HDA $* \to X \to\ !S$, $* \to Y \to\ !S$ are *bisimilar* if there is $* \to Z \to\ !S \in \mathsf{LHDA}$ and a span of open maps $X \leftarrow Z \to Y$ in LHDA.

Next we establish a correspondence between split traces [38] and cube paths in higher-dimensional tori. For us, a *split trace* over a finite or countably infinite set S is a pointed cube path in $!S$. Hence *e.g.* a split trace $a^+ b^+ a^- b^+ b^-$ (in the notation of [38]) corresponds to the cube path (i, a, ab, b, bb, b). Both indicate the start of an a event, followed by the start of a b event, the end of an a event, the start of a b event, and the end of a b event. Note that contrary to ST-traces [38], the split trace contains no information as to which of the two b events has terminated at the b^-.

By definition, a torus $!S$ on a finite or countably infinite set $S = \{a_1, a_2, \dots\}$ contains all n-cubes $(a_{i_1}, \dots, a_{i_n})$. Hence we have the following lemma:

Lemma 6.4 *Let $(x_1, \dots, x_m)$, $(y_1, \dots, y_m)$ be pointed cube paths in $!S$ with $x_m = y_m$. Then $(x_1, \dots, x_m) \sim (y_1, \dots, y_m)$.* □

Homotopy classes of split traces are thus determined by their endpoint and length:

Corollary 6.5 *The unfolding of a higher-dimensional torus $i : * \to\ !S \in \mathsf{HDO}$ is isomorphic to the pointed precubical set $j : * \to Y$ given as follows:*

- $Y_n = \{(x, m) \mid x \in\ !S_n, m \ge n, m \equiv n \mod 2\}$, $j = (i, 0)$
- $\delta_k^0(x, m) = (\delta_k^0 x, m-1)$, $\delta_k^1(x, m) = (\delta_k^1 x, m+1)$ □

The definitions of open maps and bisimilarity in $\mathsf{HDA_h}$ can now easily be extended to the labeled case. Again, we only need label-preserving morphisms.

Definition 6.6 The category of *labeled higher-dimensional automata up to homotopy* $\mathsf{LHDA_h}$ has as objects labeled HDA $* \to X \to\, !S$ and as morphisms pairs of precubical morphisms $(f, \sigma) : (* \to \tilde X \to\, !\tilde S) \to (* \to \tilde Y \to\, !\tilde T)$ of unfoldings.

Definition 6.7 A morphism $(f, \mathrm{id}) : (* \to X \to\, !S) \to (* \to Y \to\, !S)$ in $\mathsf{LHDA_h}$ is *open* if its component f is open in $\mathsf{HDA_h}$. Labeled HDA $* \to X \to\, !S$, $* \to Y \to\, !S$ are *homotopy bisimilar* if there is $* \to Z \to\, !S \in \mathsf{LHDA_h}$ and a span of open maps $X \leftarrow Z \to Y$ in $\mathsf{LHDA_h}$.

The proof of the next theorem is exactly the same as the one for Theorem 5.7.

Theorem 6.8 *Labeled HDA X, Y are homotopy bisimilar if and only if they are bisimilar.* □

Also for the labeled version, we can now show that homotopy bisimilarity agrees with history-preserving bisimilarity. We first recall the definition from [38], where we extend the labeling morphisms to cube paths by $\lambda(x_1, \dots, x_m) = (\lambda x_1, \dots, \lambda x_m)$:

Definition 6.9 Labeled HDA $* \xrightarrow{i} X \xrightarrow{\lambda}\, !S$, $* \xrightarrow{j} Y \xrightarrow{\mu}\, !S$ are *history-preserving bisimilar* if there exists a relation R between pointed cube paths in X and pointed cube paths in Y for which $((i), (j)) \in R$, and such that for all $(\rho, \sigma) \in R$,

- $\lambda(\rho) = \mu(\sigma)$,
- for all $\rho' \sim \rho$, there exists $\sigma' \sim \sigma$ with $(\rho', \sigma') \in R$,
- for all $\sigma' \sim \sigma$, there exists $\rho' \sim \rho$ with $(\rho', \sigma') \in R$,
- for all $\rho' \sqsupseteq \rho$, there exists $\sigma' \sqsupseteq \sigma$ with $(\rho', \sigma') \in R$,
- for all $\sigma' \sqsupseteq \sigma$, there exists $\rho' \sqsupseteq \rho$ with $(\rho', \sigma') \in R$.

Theorem 6.10 *Labeled HDA X, Y are homotopy bisimilar if and only if they are history-preserving bisimilar.*

7 Conclusion

We have shown that hp-bisimilarity for HDA can be characterized by spans of open maps in the category of pointed precubical sets, or equivalently by a zig-zag relation between cubes in all dimensions. Aside from implying decidability of hp-bisimilarity for HDA, and together with the results of [38], this confirms that HDA is a natural formalism for concurrency: not only does it generalize the main models for concurrency which people have been working with, but it also is remarkably simple and natural.

One major question which remains is whether also *hereditary* hp-bisimilarity can fit into our framework. Because of its back-tracking nature, it seems that simple unfoldings of HDA are not the right tools to use; one should rather consider some form of back-unfoldings of forward-unfoldings. Given the undecidability result of [24], it seems doubtful, however, that any characterization as simple as the one we have for hp-bisimilarity can be obtained.

Another important question is how HDA relate to other models for concurrency which are not present in the spectrum presented in [38]. One major such formalism is the one of *history-dependent automata* which have been introduced by Montanari and Pistore in [28, 29] and have recently attracted attention in model learning [1, 2]. We conjecture that up to hp-bisimilarity, HDA are equivalent to history-dependent automata.

With regard to the geometric interpretation of HDA as directed topological spaces, there are two open questions related to the work laid out in the paper: In [7] we show that morphisms in HDA are open if and only if their geometric realizations lift pointed directed paths. This shows that there are some connections to weak factorization systems [3] here which should be explored; see [25] for a related approach.

In [8] we relate homotopy of cube paths to directed homotopy of directed paths in the geometric realization. Based on this, one should be able to prove that the geometric realization of the unfolding of a higher-dimensional automaton is the same as the universal directed covering [12] of its geometric realization and hence that morphisms in HDA_h are open if and only if their geometric realizations lift dihomotopy classes of pointed dipaths.

The precise relation of our HDA unfolding to the one for Petri nets [22, 30] and other models for concurrency should also be worked out. A starting point for this research could be the work on symmetric event structures and their relation to presheaf categories in [36].

References

[1] Fides Aarts, Faranak Heidarian, and Frits Vaandrager. A theory of history dependent abstractions for learning interface automata. In *CONCUR*, volume 7454 of *LNCS*, pages 240–255. Springer, 2012.

[2] Fides Aarts, Bengt Jonsson, and Johan Uijen. Generating models of infinite-state communication protocols using regular inference with abstraction. In *ICTSS*, volume 6435 of *LNCS*, pages 188–204. Springer, 2010.

[3] Jiří Adámek, Horst Herrlich, Jiří Rosický, and Walter Tholen. Weak factorization systems and topological functors. *Appl. Categ. Struct.*, 10(3):237–249, 2002.

[4] Marek A. Bednarczyk. *Categories of asynchronous systems.* PhD thesis, Univ. of Sussex, 1987.

[5] Eike Best, Raymond R. Devillers, Astrid Kiehn, and Lucia Pomello. Concurrent bisimulations in Petri nets. *Acta Inf.*, 28(3):231–264, 1991.

[6] Pierpaolo Degano, Rocco De Nicola, and Ugo Montanari. Partial orderings descriptions and observations of nondeterministic concurrent processes. In Jaco W. de Bakker, Willem P. de Roever, and Grzegorz Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *LNCS*, pages 438–466. Springer, 1989.

[7] Uli Fahrenberg. A category of higher-dimensional automata. In *FOSSACS*, volume 3441 of *LNCS*, pages 187–201. Springer, 2005.

[8] Uli Fahrenberg. *Higher-Dimensional Automata from a Topological Viewpoint.* PhD thesis, Aalborg University, Denmark, 2005.

[9] Uli Fahrenberg and Axel Legay. History-preserving bisimilarity for higher-dimensional automata via open maps. *CoRR*, abs/1209.4927, 2012. http://arxiv.org/abs/1209.4927.

[10] Lisbeth Fajstrup. Dipaths and dihomotopies in a cubical complex. *Adv. Appl. Math.*, 35(2):188–206, 2005.

[11] Lisbeth Fajstrup, Martin Raussen, and Éric Goubault. Algebraic topology and concurrency. *Theor. Comput. Sci.*, 357(1-3):241–278, 2006.

[12] Lisbeth Fajstrup and Jiří Rosický. A convenient category for directed homotopy. *Theor. Appl. Cat.*, 21:7–20, 2008.

[13] Sibylle B. Fröschle and Thomas T. Hildebrandt. On plain and hereditary history-preserving bisimulation. In *MFCS*, volume 1672 of *LNCS*, pages 354–365. Springer, 1999.

[14] Philippe Gaucher. Homotopy invariants of higher dimensional categories and concurrency in computer science. *Math. Struct. Comput. Sci.*, 10(4):481–524, 2000.

[15] Philippe Gaucher. About the globular homology of higher dimensional automata. *Cah. Top. Géom. Diff. Cat.*, 43(2):107–156, 2002.

[16] Philippe Gaucher. Homotopical interpretation of globular complex by multipointed d-space. *Theor. Appl. Cat.*, 22:588–621, 2009.

[17] Philippe Gaucher. Towards a homotopy theory of higher dimensional transition systems. *Theor. Appl. Cat.*, 25:295–341, 2011.

[18] Éric Goubault. Geometry and concurrency: A user's guide. *Math. Struct. Comput. Sci.*, 10(4):411–425, 2000.

[19] Éric Goubault. Labelled cubical sets and asynchronous transition systems: an adjunction. In *CMCIM*, 2002.

[20] Éric Goubault and Thomas P. Jensen. Homology of higher dimensional automata. In Rance Cleaveland, editor, *CONCUR*, volume 630 of *LNCS*, pages 254–268. Springer, 1992.

[21] Éric Goubault and Samuel Mimram. Formal relationships between geometrical and classical models for concurrency. *Electr. Notes Theor. Comput. Sci.*, 283:77–109, 2012.

[22] Jonathan Hayman and Glynn Winskel. The unfolding of general Petri nets. In *FSTTCS*, volume 2 of *LIPIcs*, pages 223–234. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2008.

[23] André Joyal, Mogens Nielsen, and Glynn Winskel. Bisimulation from open maps. *Inf. Comp.*, 127(2):164–185, 1996.

[24] Marcin Jurdziński, Mogens Nielsen, and Jiří Srba. Undecidability of domino games and hhp-bisimilarity. *Inf. Comp.*, 184(2):343–368, 2003.

[25] Alexander Kurz and Jiří Rosický. Weak factorizations, fractions and homotopies. *Appl. Categ. Struct.*, 13(2):141–160, 2005.

[26] Antoni W. Mazurkiewicz. Concurrent program schemes and their interpretations. DAIMI Report PB 78, Aarhus University, Denmark, 1977.

[27] Robin Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[28] Ugo Montanari and Marco Pistore. An introduction to history dependent automata. *Electr. Notes Theor. Comput. Sci.*, 10:170–188, 1997.

[29] Ugo Montanari and Marco Pistore. Minimal transition systems for history-preserving bisimulation. In *STACS*, volume 1200 of *LNCS*, pages 413–425. Springer, 1997.

[30] Mogens Nielsen, Gordon D. Plotkin, and Glynn Winskel. Petri nets, event structures and domains, part I. *Theor. Comput. Sci.*, 13:85–108, 1981.

[31] David M.R. Park. Concurrency and automata on infinite sequences. In *TCS*, volume 104 of *LNCS*, pages 167–183. Springer, 1981.

[32] Carl A. Petri. *Kommunikation mit Automaten.* Bonn: Institut fr Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962.

[33] Vaughan Pratt. Modeling concurrency with geometry. In *POPL*, pages 311–322. ACM Press, 1991.

[34] Alexander M. Rabinovich and Boris A. Trakhtenbrot. Behavior structures and nets. *Fund. Inf.*, 11(4):357–403, 1988.

[35] Mike W. Shields. Concurrent machines. *Comp. J.*, 28(5):449–465, 1985.

[36] Sam Staton and Glynn Winskel. On the expressivity of symmetry in event structures. In *LICS*, pages 392–401. IEEE Computer Society, 2010.

[37] Rob J. van Glabbeek. Bisimulations for higher dimensional automata. Email message, June 1991. http://theory.stanford.edu/~rvg/hda.

[38] Rob J. van Glabbeek. On the expressiveness of higher dimensional automata. *Theor. Comput. Sci.*, 356(3):265–290, 2006.

[39] Rob J. van Glabbeek and Ursula Goltz. Equivalence notions for concurrent systems and refinement of actions. In *MFCS*, volume 379 of *LNCS*, pages 237–248. Springer, 1989.

[40] Glynn Winskel and Mogens Nielsen. Models for concurrency. In Samson Abramsky, Dov M. Gabbay, and Thomas S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 4, pages 1–148. Clarendon Press, Oxford, 1995.

A Geometric View of Partial Order Reduction

Eric Goubault Tobias Heindel Samuel Mimram[1]

CEA, LIST
Gif-sur-Yvette, France

Abstract

Verifying that a concurrent program satisfies a given property, such as deadlock-freeness, is computationally difficult. Naive exploration techniques are facing the *state space explosion* problem: they consider an exponential number of interleavings of parallel threads (relative to the program size). Partial order reduction is a standard method to address this difficulty. It is based on the observation that certain sets of instructions, called *persistent sets*, are not affected by other concurrent instructions and can thus always be explored first when searching for deadlocks. More recent models of concurrent processes use directed topological spaces: states are points, computations are paths, and equivalent interleavings are homotopic. This *geometric* approach applies theoretical results of algebraic topology to improve verification. Despite the very different origin of the approaches, the paper compares partial-order reduction with a construction of the geometric approach, the category of future components. The main result, which shows that the two techniques make essentially the same use of persistent transitions, is of foundational interest and aims for cross-fertilization of the two approaches to improve verification methods for concurrent programs.

Verifying concurrent programs is a computationally difficult task because one has to check that the desired safety properties are valid for *any possible scheduling* of the program and, typically, the number of schedulings is exponential in the size of the program. For instance, consider the following concurrent program, consisting of two processes in parallel, each of which is modifying the contents of two memory cells:

$$\texttt{x:=1;y:=2} \quad | \quad \texttt{y:=3;z:=4} \tag{1}$$

In the following, we assume that the execution model is sequentially consistent, so that an execution of the program (1) will interleave the instructions of the two processes and thus corresponds to one of the following six sequential programs.

$$\begin{array}{lll} \texttt{x:=1;y:=2;y:=3;z:=4} & \texttt{x:=1;y:=3;y:=2;z:=4} & \texttt{x:=1;y:=3;z:=4;y:=2} \\ \texttt{y:=3;x:=1;y:=2;z:=4} & \texttt{y:=3;x:=1;z:=4;y:=2} & \texttt{y:=3;z:=4;x:=1;y:=2} \end{array} \tag{2}$$

In order to verify that the program (1) is correct, one could thus use traditional verification techniques on the six programs (2), which can also be pictured as the paths from x_{00} to x_{22} on the left of Figure 1. However, this approach will not scale

[1] The authors gratefully acknowledge support from the project PANDA ANR-09-BLAN-0169.

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.com/locate/entcs

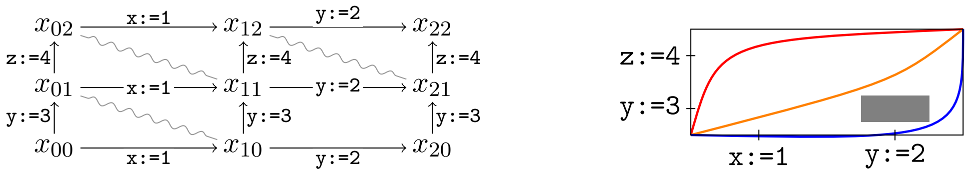


Fig. 1. Asynchronous transition semantics and geometric model of (1).

up because the number of sequential programs corresponding to a concurrent one grows very fast: this phenomenon is called *state space explosion*. In order to avoid it, techniques of *state space reduction* have been invented that exploit the inherent dependencies between instructions. For instance, the order in which `x:=1` and `y:=3` are executed is not relevant for most program properties: both possible executions result in the same memory state in the end. We use *tiles* (marked by ∿∿) on the left of Figure 1 to indicate when instructions can be switched in this way, and say that the instructions commute. Formally, the graph with tiles forms an *asynchronous transition system* or ATS for short [19].

Notice that, in this program, the instruction `x:=1` is *persistent* in the sense that it commutes with all possible instructions running in parallel with it (forming the second process). Without loss of generality, we might thus suppose that it is executed first and we are left to verify that the program `x:=1;(y:=2|y:=3;z:=4)` is valid. This program gives rise to only three possible executions, compared to the six of (2): we can avoid examining the paths going through the states x_{01} and x_{02}. Of course we can iteratively use this procedure to reduce the program further, which results in removing paths going through the state x_{12}. This procedure, introduced by Valmari and developed by Godefroid [7], is called *partial order reduction* (POR) and has lead to a wide variety of successful tools such as SPIN [14] (where the above mentioned persistent sets are complemented with other techniques such as sleep sets). These tools have been originally devised to optimize deadlock detection (and have been extended afterwards in various ways). Thus, we shall focus on deadlock detection, though the intended range of application is wider (full reachability will be detailed in future works); moreover, we shall mainly be concerned with acyclic systems, which is also a common restriction in techniques of partial order reduction (see for example [6]).

Independently of advances of POR, work on *topological semantics* of concurrent programs has lead to new techniques based on similar observations, but formulated in a much different context [8,9]. In this line of research, concurrent programs are modeled as directed topological spaces [4]. For instance, on the right of Figure 1, we see the space associated to the program (1): ignoring the curvy paths at this point, the space is essentially a filled square with a square hole (rendered in gray). Notice the strong resemblance between the space and the asynchronous transition system on the left: in some precise sense, the topological model is actually the geometrical counterpart of the ATS (*viz*, its *geometric realization*): constructions in one setting can be reformulated in the other; avoiding the formal details about topology, we appeal to geometric intuition to illustrate the main ideas.

In the geometric approach, program executions are modeled as paths in a space, as shown in Figure 1. For instance the path that passes underneath the hole corresponds to the first possible scheduling in (2). Notice that paths should always be monotone when projected to a coordinate; this captures progression in time of each thread of a program. This is why topological spaces of the model need to be endowed with a notion of *direction* [13]. In these topological models, two interleavings that can be obtained from each other by repeatedly switching commuting instructions are represented as *dihomotopic* paths, i.e. paths which can be continuously deformed into each other. For instance, on the right of Figure 1, the two paths that pass on top of the hole are dihomotopic, but none of them is dihomotopic to the one that passes underneath; the reason is that the hole is an obstacle to continuous deformation of paths below and above the hole.

The interest of defining a program semantics in terms of topological spaces, is that it allows one to reuse concepts and tools coming from algebraic topology. This has enabled the formulation of state space reduction methods as follows. From a concurrency point of view, a path is *inessential* if it does not change the possible future paths, up to homotopy: such a path corresponds to an execution where neither the program has made a significant choice (such as choosing a branch of a conditional choice) nor the scheduler (such as ordering two concurrent actions which do not commute). Using a suitable formalization, one is naturally lead to consider the category of paths in the topological semantics of a program, quotiented by inessential paths in order to only retain the structure which is relevant for studying the program up to commutation of instructions, i.e. define a notion of reduced state space. This category, introduced in [11], is called the *category of future components* and is used in the tool ALCOOL [4,9], which has been successfully used in an industrial context [2] for deadlock detection.

Even though there is a striking similarity between the notions of persistent sets and inessential paths, the relationship between the two has never been formally studied and the purpose of this article is to fill this gap, in order for partial order reduction and geometric techniques to improve each other and combine their potential to alleviate the state space explosion problem. As a result, we are able to show that, under fairly reasonable assumptions, persistent transitions are the algebraic counterpart of inessential paths. Despite the fact that the analogy is intuitive, it turns out that the theoretical comparison is sometimes technically involved. On a more practical side, a preliminary comparison, based on experiments, was started in [4].

In Section 1, we begin with a review of the models of computations used here to formalize persistent sets: *labeled transitions systems with independence* (LTSI), which are generalized into *asynchronous transitions systems* (ATS). In Section 2, we conservatively extend the original definition of persistent sets from LTSI to ATS, which are closer to geometric models, and show that they retain their fundamental reduction potential to prune the search space for deadlock detection. Finally, we develop in Section 3 a first conceptual link between partial order reduction and directed algebraic topology; in Theorem 3.10, we make precise in what sense *persistent singletons* are essentially the same as *inessential morphisms*, thus identifying a common concept of geometrical and partial order reduction methods (amongst the host of techniques and heuristics that are used in both approaches). This is further discussed in our summary in Section 4, together with venues for future research.

1 Models of concurrent computation

We shall use two models for concurrent computations: labeled transitions systems with independence (LTSI), as traditionally used in the POR technique, and asynchronous transition systems (ATS), which can be seen as algebraic counterparts of the directed topological spaces of the geometric point of view. Both of these formalize the state spaces of programs, such as the example given in Figure 1.

1.1 Labelled transition systems with independence

A labeled transition system (LTS) is a triple $(S, \Lambda, \to)$ where S is a set of *states*, Λ is a set of *transition labels*, and $\to \subseteq S \times \Lambda \times S$ is a *transition relation*. We write $s \xrightarrow{t} s'$ whenever $(s, t, s') \in \to$ and $s \xrightarrow{t}$ when $t \in \Lambda$ is *enabled* in s, i.e. when there exists a state s' such that $s \xrightarrow{t} s'$. We shall here consider only deterministic transition systems, i.e. $s \xrightarrow{t} s'$ and $s \xrightarrow{t} s''$ imply $s' = s''$.

Two (labels of) transitions are considered to be independent if the relative order in which they are performed does not matter, in the sense that both schedulings are essentially the same computation and in particular yield the same result. This notion of independence motivates the following definition [7].

Definition 1.1 (Independence) *A* labeled transition system with independence (LTSI) *consists of an* LTS $(S, \Lambda, \to)$ *with an* independence relation $\parallel$, *which is a symmetric, irreflexive relation* $\parallel \subseteq \Lambda \times \Lambda$ *such that for each pair* $(t_1, t_2) \in \parallel$ *and every reachable state* s,

(i) *if* $s \xrightarrow{t_1} s'$ *then* $s' \xrightarrow{t_2}$ *iff* $s \xrightarrow{t_2}$*; and*

(ii) *if both* $s \xrightarrow{t_1}$ *and* $s \xrightarrow{t_2}$ *hold, there exists a unique* s'' *such that (for a unique pair of states* $s_1, s_2 \in S$*) both* $s \xrightarrow{t_1} s_1 \xrightarrow{t_2} s''$ *and* $s \xrightarrow{t_2} s_2 \xrightarrow{t_1} s''$ *hold.*

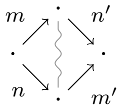
1.2 Asynchronous transition systems

Geometric models for concurrency have been introduced with the idea to apply tools developed in algebraic topology to the analysis of concurrent programs [8]; those programs are considered as directed topological spaces, in which (directed) paths correspond to particular executions of the program, and dihomotopy between paths is an equivalence of executions (relating schedulings of a concurrent program that lead to the same result). For the analysis of concurrent systems, the topological formalization can be replaced by its algebraic counterpart, namely asynchronous transition systems (or more generally precubical sets), which brings us closer to the formalism of LTSI as we recall here, see [12] for a detailed comparison between those models.

Recall that a *graph* $G = (V, E, \mathrm{src}, \mathrm{tgt})$ consists of a set V of *vertices*, a set E of *edges*, and two functions $\mathrm{src}, \mathrm{tgt} \colon E \to V$, which associate to each edge its *source* and *target*, respectively. A *path* u from vertex x to vertex y, denoted by $u \colon x \twoheadrightarrow y$, is a sequence of consecutive edges; the empty path on a vertex x is denoted by $\varepsilon_x \colon x \twoheadrightarrow x$. The concatenation of two paths $u \colon x \twoheadrightarrow y$ and $v \colon y \twoheadrightarrow z$ is denoted by $u \cdot v \colon x \twoheadrightarrow z$.

Definition 1.2 (Asynchronous transition system) *An* asynchronous transition system*, or* ATS *for short, is a pair $(G, \diamond)$ where G is a graph (whose vertices are called* positions *and edges are called* transitions*) and $\diamond$ is a relation on the set of paths of length two that have the same source and target, i.e. $(u, v) \in \diamond$ only if $u, v \colon x \twoheadrightarrow y$. The elements of $\diamond$ are called* tiles*. Two transitions $m \colon x \to y$ and $n \colon x \to z$ are* independent*, written $m \parallel n$, if there exist transitions m' and n' such that $(m \cdot n') \diamond (n \cdot m')$, i.e. we have the tile as in (3), on the right below.*

To formalize the fact that we can reschedule independent events in the run of a concurrent system without changing the actual computation that is performed, we use the following definition of trace, which refines Mazurkiewicz's notion and is the algebraic counterpart of homotopy in topological spaces.



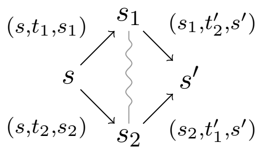
(3)

Definition 1.3 (Trace) *Two paths $u, v \colon x \twoheadrightarrow y$ are* homotopic*, written $u \sim v$, if u can be obtained from v by repeatedly replacing path fragments $m \cdot n'$ by $m' \cdot n$ whenever there exists a tile (3), i.e. the relation $\sim$ is the smallest congruence w.r.t. path composition that extends $\diamond$. A* trace *is an equivalence class w.r.t. $\sim$ and we write $[u]$ for the equivalence class of a path u.*

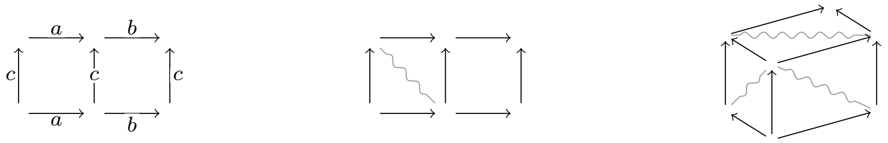
Definition 1.4 (Trace category) *The* trace category *associated to an* ATS *$(G, \diamond)$ has the vertices of G as objects and traces $[u]$ with $u : x \twoheadrightarrow y$ as morphisms from x to y. Composition is the operation induced on traces by path concatenation and identities are empty traces $[\varepsilon_x]$.*

In order to relate this model with the one introduced in previous section, we should remark that each LTSI (Definition 1.1) naturally induces an ATS as follows.

Definition 1.5 (Induced ATS) *Let $(\mathbb{S}, \parallel)$ be an* LTSI *where $\mathbb{S} = (S, \Lambda, \to)$. Its* induced ATS*, denoted by $\text{ATS}^{\parallel}_{\mathbb{S}} = (G, \diamond)$, has vertices as positions and edges as transitions, i.e. $V = S$ and $E = \to$; moreover, for each $e = (s, t, s') \in E$ we have $\text{src}(e) = s$, $\text{tgt}(e) = s'$, and $\diamond$ contains a tile as pictured on the right whenever $t_1 \parallel t_2$ and $s \xrightarrow{t_1}$ and $s \xrightarrow{t_2}$.*



Example 1.6 Consider the LTSI on the left below with $\parallel = \{(a, c), (c, a)\}$; its associated ATS is shown in the middle. Notice that no LTSI can generate the ATS on the right, because to generate its transitions, we would have to generate the "full cube" as stated in Lemma 2.11. In this sense, ATS are more general than LTSI.



A labeled variant of ATS can easily be defined and all the constructions performed on ATS in this article can be generalized to the labeled case (see [12] for the precise relations between those models). However, the labels of transitions are not really needed since a suitable notion of transition label can be recovered abstractly as its associated event:

Definition 1.7 (Event) *Let $(G, \diamond)$ be an* ATS*. Two transitions m and m' are* parallel*, written $m \,\square\, m'$, if they form opposite sides of some tile (3), i.e. if there exists a tile $(m \cdot n', n \cdot m') \in \diamond$ for some transitions n and n'. An* event *is an equivalence class w.r.t. the least equivalence relation on transitions that contains $\square$.*

For instance, in the middle ATS in Example 1.6, the two vertical transitions on the left are elements of the same event, while the rightmost is not: considering the LTSI on the left, since b and c are not independent, performing c before or after b does not correspond to the same event – even though they carry the same label.

Remark 1.8 In the sequel, we shall restrict to ATS that have unique ways to "close" tiles, and "switchings" of transitions are uniquely defined. Formally, the former means that $(m \cdot p) \diamond (n \cdot q)$ and $(m \cdot p') \diamond (n \cdot q')$ imply $p = p'$ and $q = q'$ while the latter says that $u \diamond v$ and $u \diamond w$ imply $v = w$. All ATS that are induced by LTSI satisfy this property.

2 Persistent sets in asynchronous transition systems

Persistent sets are one of the most well-known techniques to reduce the number of executions to be explored to check that a concurrent program cannot lead to a deadlock state. Here, we generalize their definition to ATS to compare it with geometric reduction techniques in Section 3. We begin by recalling Godefroid's original definition [7].

Definition 2.1 (Persistent set) *Given an* LTSI $(S, \Lambda, \to, \|)$ *and a state $s \in S$, a set $R \subseteq \Lambda$ of transition labels that are enabled in a state $s \in S$ is* persistent *in s, if for all nonempty transition sequences*

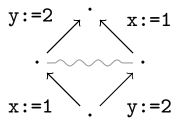
$$s = s_1 \xrightarrow{t_1} s_2 \xrightarrow{t_2} s_3 \ldots \xrightarrow{t_{n-1}} s_n \xrightarrow{t_n} s_{n+1}$$

from s that satisfy $t_i \notin R$ $(1 \leq i \leq n)$, $t_n \parallel t$ holds for every transition $t \in R$.

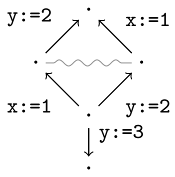
A single transition can be considered persistent whenever it can be pulled back to the state at which it was first enabled by permuting it with independent (not necessarily persistent) transitions. More generally, persistent sets typically comprise the actions of a conditional choice of some concurrent component, as illustrated in the following example.

Example 2.2 Consider the programs below, with their respective LTSI semantics.

```
x:=1 | y:=2
```



```
if  (z<0)
    x:=1 | y:=2
else
    y:=3
```



(4)

In the LTSI on the left, we have `x:=1` and `y:=2` running completely in parallel; thus, both {`x:=1`} and {`y:=2`} are persistent sets at the initial state. In contrast, in the system on the right, the transitions `x:=1` and `y:=2` are in conflict with `y:=3`; the only persistent set consists of a "monolithic" component with no threads running concurrently, i.e. {`x:=1`, `y:=2`, `y:=3`} is the only persistent set at the initial state. As a final example, in Figure 1, {`x:=1`} and {`x:=1`, `y:=3`} are persistent in the initial state while the set {`y:=3`} is not.

Recall that a *deadlock* is a state in which no transition is enabled. As explained before, the main interest of persistent sets is to narrow the search for deadlocks in a concurrent program: given a choice of persistent set for each state, a reachable deadlock is always reachable by a path containing only transitions in the chosen persistent sets; it is therefore sufficient to explore a system "along" persistent sets:

Proposition 2.3 (Persistent deadlock reachability [7, Theorem 4.3]) *Given a choice of a non-empty persistent set R_s at each state s that is not a deadlock, for each path $s_0 \xrightarrow{t_0} s_1 \dots s_n \xrightarrow{t_n} d$ to a deadlock d, there exists a path $s_0 \xrightarrow{t'_0} s'_1 \dots s'_n \xrightarrow{t'_n} d$ such that $t'_i \in R_{s_i}$ for all $i \in \{0, \dots, n\}$.*

The proof of the preceding proposition is based on the following observation: for each path $u : s \twoheadrightarrow d$ leading to a deadlock d and persistent set R at s, there exists a path $v \in [u]$ (i.e. $v \sim u$) whose initial transition is in R. This motivates our generalization of the definition of persistent sets, based on the following definitions:

Definition 2.4 *Let $(G, \diamond)$ be an* ATS*. Given a path $u \colon x \twoheadrightarrow z$, a transition $m \colon x \to y$ is* initial modulo homotopy *if there exists a path $v \colon y \twoheadrightarrow z$ such that $u \sim m \cdot v$; the set of all transitions that are initial modulo homotopy in u is denoted by $\tilde{\imath}(u)$. Two paths $u \colon x \twoheadrightarrow y$ and $v \colon x \twoheadrightarrow z$ with common source x are* compatible*, written $u \uparrow v$, if there exist paths $w_y \colon y \twoheadrightarrow x'$ and $w_z \colon z \twoheadrightarrow x'$ with common target x' such that $u \cdot w_y \sim v \cdot w_z$.*

Thus, in particular all transitions that are initial modulo homotopy in some path are pairwise compatible. These notions enable us to generalize persistence to ATS as follows:

Definition 2.5 (Homotopy persistent set) *Let R be a set of transitions in an* ATS *$(G, \diamond)$ that share a state x as common source. The set R is* homotopy persistent*, if each path $u \colon x \twoheadrightarrow z$ is compatible with all transitions in R provided that no transition in R is amongst its homotopy initial ones, i.e.*

$$\forall u \colon x \twoheadrightarrow z, \qquad R \cap \tilde{\imath}(u) = \varnothing \qquad \Rightarrow \qquad \forall m \in R.\ u \uparrow m.$$

Remark 2.6 A *persistent singleton* (a persistent set with a single element) corresponds to a transition that is compatible with all paths with the same source.

Example 2.7 The singleton $\{o \to y\}$ is an homotopy persistent set in the ATS below in (5). Note that the transition $o \to y$ is compatible with the transition $o \to x$ even though the two transitions are not "independent".

Situations like this typically arise in consumer producer problems where n-ary semaphores are used to ensure that an exhausted resource is not used, such as when implementing a queue with limited size. For instance, suppose that we have a queue in which we can `put` at most two elements (if there are already two

$$\begin{tikzcd}
z & y' \arrow[l] & & \bar{y}' \arrow[r] \arrow[dr, squiggly, no head] & \bar{z} \\
x' \arrow[u] \arrow[ur, squiggly, no head] & o' \arrow[l] \arrow[u] \arrow[r, squiggly, no head] & y \arrow[ul] \arrow[ur] \arrow[r, squiggly, no head] & \bar{o}' \arrow[u] \arrow[r] & \bar{x}' \arrow[u] \\
& x \arrow[ul] \arrow[u, squiggly, no head] & o \arrow[l] \arrow[ul] \arrow[u] \arrow[ur] \arrow[r] & \bar{x} \arrow[ur] \arrow[u, squiggly, no head] &
\end{tikzcd} \tag{5}$$

elements, the `put` operation blocks until an element is `take`n from the queue).

The following program generates the above ATS (the arrow subscripts indicate the direction of the corresponding transitions).

```
put↑   |   if (...) {take↗ | put→} else {take↖ | put←}
```

It remains to show that the notion of homotopy persistent set is in fact a conservative extension of the original one (Definition 2.1). The proof is based on the observation that in each ATS that is induced by an LTSI, a transition $m : x \to y$ has a unique residual path $u/m : y \twoheadrightarrow z'$ after a compatible path $u : x \twoheadrightarrow z$; we say this kind of ATS *has compatible residuals.* The residual u/m of m after u has intuitively the "same effect" as u, once m has been performed. The assumption made in Remark 1.8 is necessary for the following definition to be sound.

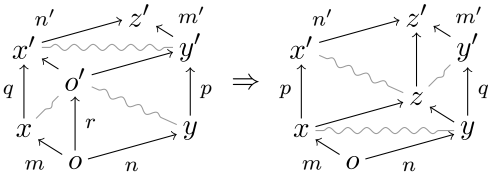
Definition 2.8 (Residual) *Let $m\colon x \to y$ be a transition, let $u : x \twoheadrightarrow z$ be a path. The* residual *of u after m, denoted by u/m, and the* residual *of m after u, denoted by m/u, are defined by induction on the length of u as follows.*

- $\varepsilon_x/m = \varepsilon_y$ *and* $m/\varepsilon_x = m$
- $(m \cdot u')/m = u'$ *and* $m/(m \cdot u') = \varepsilon$
- *If $n \neq m$, $(n \cdot u)/m = n' \cdot (u/m')$ where m' and n' are transitions such that $m \cdot n' \diamond n \cdot m'$ as in (3) (where m' is uniquely determined by Remark 1.8).*

$$\begin{tikzcd}
y_1 \arrow[r] \arrow[rrrr, two heads, shift left=2, "u/m"] \arrow[rrrrrrrr, bend left, "v/m"] \arrow[dr, squiggly, dash] & y_2 \arrow[r] \arrow[dr, squiggly, dash] & y_3 & \cdots y_{k-1} \arrow[r] \arrow[dr, squiggly, dash] & y_k \arrow[r, two heads, "\varepsilon"] & y_k \arrow[r] & y_{k+1}\cdots & y_{l-1} \arrow[r] & y_l \\
x_1 \arrow[u, "m"] \arrow[r] \arrow[rrrr, two heads, shift right=2, "u"'] \arrow[rrrrrrrr, bend right, "v"'] & x_2 \arrow[u] \arrow[r] & x_3 \arrow[u] & \cdots x_{k-1} \arrow[u] \arrow[r] & x_k \arrow[u, "m/u"'] \arrow[r, "m/u"'] & y_k \arrow[u, two heads, "\varepsilon"] \arrow[r] & y_{k+1}\cdots \arrow[u, two heads, "\varepsilon"] & y_{l-1} \arrow[u, two heads, "\varepsilon"] \arrow[r] & y_l \arrow[u, two heads, "\varepsilon", "m/v"']
\end{tikzcd}$$

Remark 2.9 The residual m/u of a transition m after a path u (when it exists) is either a transition or empty (as illustrated above). It is straightforward to extend the definition to the residual v/u of a path $v\colon x \twoheadrightarrow z$ after a path $u\colon x \twoheadrightarrow y$.

The fact that every ATS that is induced by an LTSI has compatible residuals can be deduced from the fact that they satisfy a particular diagrammatic property; it can be expressed as follows [17].



(6)

Definition 2.10 (Forward Cube Property)
An ATS has the Forward Cube Property *(or FCP) if for every three tiles as shown on the left in (6) there exist three matching tiles as shown on the right in (6).*

Lemma 2.11 *The induced ATS of an LTSI has the FCP property.*

Our main interest in this property is the following property [16]:

Proposition 2.12 (Residuation) *An ATS with the FCP has compatible residuals.*

This proposition is the main tool to show that in fact our definition of persistent set is a conservative extension of the original one:

Proposition 2.13 (Homotopy persistent is persistent) *Let $(S, \Lambda, \rightarrow, \|)$ be an* LTSI*, let x be a position, and let $R \subseteq \Lambda$ be a set of transitions that are all enabled at x. The set R is persistent at x if and only if the set $R' = \{(x, t, y) \mid t \in R, x \xrightarrow{t} y\}$ is homotopy persistent at x in* $\mathrm{ATS}^{\|}_{\mathbb{S}}$.

Proof. If R is persistent, it is easy to show that R' is homotopy persistent since if a transition label in R is independent with all transitions labels on the path, it is in particular compatible with the corresponding transitions (by Definition 2.8). Conversely, assume that R' is homotopy persistent. Since $\mathrm{ATS}^{\|}_{\mathbb{S}}$ has the FCP by Lemma 2.11, for every path $u\colon x \twoheadrightarrow z$, we can use Proposition 2.12 to show that either some residual of a transition in R' occurs in u (if $\tilde{\imath}(u) \cap R' \neq \varnothing$), or we have residuals of all transitions in R' after u (if u is compatible with all transitions in R'). It can easily be checked that the residual of any transition in R' always carries the same label as the "original" in R'. □

With this result at hand, we refer to homotopy persistent sets in ATS as persistent ones from now on. Moreover, we have a further successful "soundness check" for our generalization of persistent sets, namely, the fundamental fact about reachability of deadlocks "along" persistent sets, namely Proposition 2.3, lifts to any ATS.

Lemma 2.14 *Let $G = (G, \diamond)$ be an* ATS*, let $u\colon x \twoheadrightarrow d$ be a path such that d is a deadlock and let R be a non-empty persistent set at x. Then R contains some of the initial transitions of u (i.e. $u \sim m \cdot v$ for some $m \in R$ and a suitable path v).*

Proof. Consider a path $u\colon x \twoheadrightarrow d$ leading to a deadlock d. Suppose that u is compatible with all transitions in R. Because R is non-empty, there exists some transition $m\colon x \rightarrow y$ in R and paths $u'\colon d \twoheadrightarrow z$ and $v\colon y \twoheadrightarrow z$ such that $u \cdot u' \sim m \cdot v$. Since d is supposed to be a deadlock, we necessarily have $u' = \varepsilon_d$ and $z = d$. Therefore $u \sim m \cdot v$, i.e. $m \in \tilde{\imath}(u)$ and we conclude. Otherwise, u is incompatible with some transition $m\colon x \rightarrow y$ of R. Since R is homotopy persistent, $R \cap \tilde{\imath}(u) \neq \varnothing$.□

Corollary 2.15 (Persistent deadlock reachability) *Let $(G, \diamond)$ be an* ATS *with V and E as set of vertices and edges respectively, let $R_{\_}\colon V \rightarrow \wp(E)$ be a function such that for all non-deadlocking states $x \in V$, the set R_x is a non-empty persistent set at x. Given a deadlock $d \in V$ reachable by a path $u : x_0 \twoheadrightarrow d$, there exists a path $v : x_0 \twoheadrightarrow d$ such that $u \sim v$ and every transition $m : x \rightarrow y$ occurring in v is persistent at x, i.e. $m \in R_x$.*

Note that for arbitrary safety properties, persistent sets alone do not suffice, and one has to use extra techniques similar to the sleep sets of [7].

Finally, the following technical lemma will be useful in the following:

Lemma 2.16 *In the trace category associated to an* ATS *which satisfies the* FCP *every morphism is epi, that is for every paths $u : x \twoheadrightarrow y$ and $v, w : y \twoheadrightarrow z$, $[u \cdot v] = [u \cdot w]$ implies $[v] = [w]$.*

3 Comparing POR and categories of future components

In this section, we relate the notion of persistent set with the construction of the *category of future components* [10], which gives a condensed representation of (geometric models of) concurrent programs by eliminating states that enable only "inessential" transitions. We first reformulate this construction in the setting of ATS, as well as related properties: we introduce the notion of future-reflecting trace, and the category of future components is then defined as the quotient of the category of traces by a consistent set of future-reflecting traces. After that, the crucial point is to make precise which transitions are considered as inessential. Intuitively, one might expect that all persistent transitions are inessential. However, the general definition of ATS allows peculiar situations which do not occur in ATS that are generated by usual programs (for instance persistent transitions might not be stable under residuation). Nevertheless, we will show, for suitably "well-behaved" ATS, that inessential transitions are the same as persistent ones and that the category of future components does only contain states that do not enable persistent transitions.

The first requirement for *inessential* transitions is that they do not influence any choices that might lead to deadlocks. Intuitively, choices available in the future before and after performing an inessential transition should be the same: they should be future reflecting in the following sense.

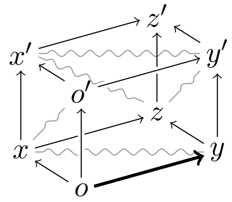
Definition 3.1 (Future-reflecting trace) *A trace $[u]\colon x \twoheadrightarrow y$ is* future-reflecting *if for each position z reachable from y (i.e. there exists a path $y \twoheadrightarrow z$), precomposition with $[u]$ induces a bijection between traces from y to z and traces from x to z.*

$$\begin{tikzcd}
 & & z & & & & z & & & & z \\
x \arrow[r, "{[u]}"'] & y \arrow[ur, dashed] & & \Rightarrow & x \arrow[r, "{[u]}"'] & y \arrow[ur, "{[v]}"'] & & \leftrightsquigarrow & x \arrow[r, "{[u]}"'] \arrow[urr, "{[v']}"] & y &
\end{tikzcd}$$

Example 3.2 (Future-reflecting transition) Reconsider the ATS in Figure 1. The only important choice for scheduling the transitions concerns the relative order of the instructions `y:=3` and `y:=2`. Namely, the transition $x_{00} \to x_{10}$ (with label `x:=1`) does not influence the choice and thus is intuitively "inessential". In fact, it reflects all futures according to the definition. For example, the two traces from x_{10} to x_{22} factor uniquely through $x_{00} \to x_{10}$. In contrast, the transition $x_{10} \to x_{20}$ (with label `y:=2`) does not reflect futures as there is only one trace from x_{20} to x_{22} while there are two traces from x_{10} to x_{22}.

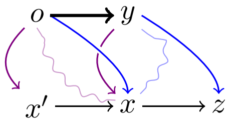
In other words, if a trace is future-reflecting, all choices in the future are already present at their source. The notion of future-reflecting *transition* is close to the notion of persistent transition; however, the two do not generally coincide as seen in the following examples.

Example 3.3 Consider the cube on the right. All pairs of transitions are independent except for $o \to o'$ and $o \to y$ (the front face of the cube is not a tile). Now, both transitions $o \to y$ and $o \to o'$ are persistent since they are compatible with all other transitions from o. However



neither of them is a future-reflecting trace. To see that $o \to y$ is not a future-reflecting trace, consider the trace $[y \to y']$. There is only one trace from y to y' but two from o to y'. The argument for $o \to o'$ is symmetric. The important point to notice in this example, is that the associated trace category is not a poset (it might have more than one morphism between two objects). In fact, it can be shown that when the trace category is a poset (this is the case for event structures), all persistent transitions are future-reflecting.

In the ATS on the right, the transition $o \to y$ is persistent. However, it is not a future-reflecting trace since the trace $[o \to x]$ does not factor through $[o \to y]$. Also note that this ATS is actually induced by an LTSI.



The basic idea of state space reduction used in the category of future components is that future-reflecting transitions are not informative from a concurrency point of view and thus need not be represented explicitly. So, starting from an ATS, one might be tempted to consider the associated trace category (Definition 1.4) and quotient it by all the future-reflecting traces, which amounts to formally turn them into identities. It turns out that this crushes too much information about traces (see [5], in particular there is no equivalence between the quotient and the fraction category and no compositionality result via Van Kampen theorems). A suitable solution is to quotient wrt a subset of all future-reflecting traces, namely those that are closed under composition and "residuals"; the formal details are as follows.

Definition 3.4 (System of inessentials) *Let $(G, \diamond)$ be an* ATS, *and let Σ_f be a set of traces. The set Σ_f is a* system of inessentials (SOI) *if*

(i) *each element of Σ_f is a future-reflecting trace;*

(ii) *Σ_f contains all empty traces and is closed under composition;*

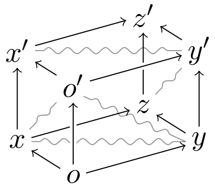
(iii) *Σ_f is stable under pushout, i.e. for every trace $\sigma\colon x \twoheadrightarrow z \in \Sigma_f$ and for any trace $[u]\colon x \twoheadrightarrow y$, there exists a pushout $z \xrightarrow{[u']} x' \xleftarrow{\sigma'} y$ of $z \xleftarrow{\sigma} x \xrightarrow{[u]} y$ such that $\sigma' \in \Sigma_f$.*

We recall that the *pushout* of two morphisms $f : A \to B$ and $g : A \to C$ in a category $\mathcal{C}$ is a pair of morphisms $g'\colon B \to D$ and $f'\colon C \to D$ such that $f' \circ g = g' \circ f$ and moreover for any other pair of morphisms $h\colon B \to D'$ and $k\colon C \to D'$ that satisfy $k \circ g = h \circ f$, there exists a unique morphism $l\colon D \to D'$ such that both $k = l \circ f'$ and $h = l \circ g'$ (as illustrated to the right).

$$\begin{tikzcd}
& B \arrow[dr, "g'"] \arrow[rr, bend left, "h"] & & \\
A \arrow[ur, "f"] \arrow[dr, "g"'] & & D \arrow[r, dashed, "l"] & D' \\
& C \arrow[ur, "f'"] \arrow[rru, bend right, "k"'] & &
\end{tikzcd}$$

Definition 3.5 (Inessential trace) *A trace is* inessential *if it belongs to the union of all systems of inessentials of an* ATS *(which is a non-empty* SOI *which is maximal wrt inclusion).*

In fact, in most of the following examples, the pushout of an inessential trace along another one is just its residual. The following example shows that the set of future-reflecting transitions need not be closed under residuals and thus the maximal SOI does not contain all future-reflecting traces.



Example 3.6 Consider the ATS on the right, in which all faces but the back face are tiles. The transition $o \to o'$ is future-reflecting. Its residual after $o \to x$, namely $x \to x'$, is not future-reflecting (and not even persistent). In fact, the largest SOI is the closure of $\{o' \to x', o' \to y', y \to z, y \to y'\}$ by residuals, composition, and identities.

Note that the ATS of this example does not satisfy the Forward Cube Property. Nevertheless, the category of future components is well-defined.

Definition 3.7 (Category of future components) *The category of future components of an* ATS *is its trace category quotiented by inessential traces.*

This construction amounts to forgetting inessential transitions by considering them as identities. Another point of view, formalized by the following proposition, is that this construction removes states which enable inessential transitions: informally, passing through them does not bring any new information about possible future traces (as no important choice can be made by the program or the scheduler). This agrees with the informal explanations of the introductory example in Figure 1; the state space reduction removes states x_{01}, x_{02} and x_{12}.

Proposition 3.8 *Let $(G, \diamond)$ be an* ATS*, let Σ be the maximal* SOI*. If Σ is finite, the category of future components is the full subcategory of the category of traces that is induced by all states that do not enable any transition in Σ.*

Finally, we give sufficient conditions which imply that the category of future components yields the expected state space. The final condition is the absence of déjà vus (cf. Example 3.3).

Definition 3.9 (Déjà vu) *A* déjà vu *is a transition $m \colon x \to y$ such that there exists a path $u \colon y' \twoheadrightarrow x$ and a transition $m' \colon x' \to y'$ in u that is the same event as m. The* ATS *is* déjà vu free *if none of its transitions are déjà vus.*

In other words, an ATS is déjà vu free if none of its paths contains two transitions that are instances of the same event. For example, in the second ATS of Example 3.3, the transition $x \to z$ is a déjà vu since the path $o \to x' \to x \to z$ contains two occurrences of the same event: $(x \to z) \square (o \to y) \square (x \to x')$. Similarly, any cyclic ATS has déjà vus though déjà vus need not necessarily be cycles. With this final proviso, we obtain a formal correspondence between inessential and persistent transitions (i.e. transitions $m : x \to y$ such that $\{m\}$ is a persistent set at x).

Theorem 3.10 (Inessential vs. persistent transitions) *In any déjà vu free* ATS *that has the forward cube property, a transition is persistent iff it belongs to some* SOI *(and in particular the maximal one). If the trace category of the* ATS *is finite, the category of future components is the full subcategory containing all objects that do not enable persistent transitions.*

Proof. Assume that Σ_f is a SOI and $m \in \Sigma_f$ a transition. We want to show that $\{m\}$ is a persistent set. For any path coinitial with m, we have a pushout of m along u because Σ_f is closed under pushouts. Thus $m \uparrow u$ and m is persistent by Remark 2.6.

Conversely, let Σ' be the set of traces that consist only of persistent transitions.

It suffices to show that Σ' is a SOI. To show that Σ' is stable under pushouts, let $m_1 \cdots m_k \in \Sigma'$ and let u be a path coinitial with m_1. We successively take residuals of m_i along u using the FCP, Remark 2.6, and Proposition 2.12, and verify that the composite of the residuals is a pushout. Next, we show that persistent transitions are future-reflecting. Let $m\colon x \to y$ be a persistent transition and z a state reachable from y. Since the ATS is supposed to have the FCP, the transition m is an epimorphic trace by Lemma 2.16, i.e. for every paths u and v such that $[m \cdot u] = [m \cdot v]$ we have $[u] = [v]$. Namely, in this case we have $u = (m \cdot u)/m \sim (m \cdot v)/m = v$, the middle homotopy being justified by the fact that $m \cdot u \sim m \cdot v$ and residuation is compatible with homotopy [16]. Precomposition with m with traces from y to z is thus injective and it remains to show that it is surjective. Given a path $u\colon x \twoheadrightarrow z$, we have to show that u factors through m. The transition m is compatible with u by Remark 2.6, and thus we can pushout $[m]$ along $[u]$, which is just the residual of m after u; it must be the identity as we would obtain a déjà vu otherwise and thus, in fact, $[u]$ factors as $[m] \cdot [u']$ where $[u']$ is the pushout of $[u]$ along $[m]$. □

Thus, in a large class of common systems, including those generated by event structures or Petri nets with acyclic causality, persistent singletons are in fact the same as inessential morphisms. Thus, despite the huge gap between the origins of the geometric approach and the POR technique, in a large class of systems, we do not have only "obvious" similarities, but in fact, a formal argument that inessential transitions are exploited in the same way.

4 Conclusion

We have developed a conservative extension of persistent sets to asynchronous transition systems (Definition 2.5) that coincides with the original concept on LTSI (Proposition 2.13). This extension forms the base of our comparison of the partial order reduction technique and the geometric approach to state space reduction, since ATS are the algebraic counterpart of geometrical models. In particular, we have shown that our definition retains the main application of persistent sets which consists in pruning the search for deadlocks (Corollary 2.15).

These preliminaries are crucial for our main contribution, which demonstrates the practical relevance of the theoretical construction of the category of future components and further results in [10], where state space reduction is performed for general directed topological spaces. Glossing over details, Theorem 3.10 says that inessential transitions are the same as persistent singletons. As a direct consequence, the construction of the category of future components roughly amounts to the application of the POR technique when we use only persistent *singletons*. Thus, we have found a common core of the geometric approach and the POR technique, while both approaches have additional heuristics and methods to improve performance. There are favorable examples for the geometric approach [4], which motivates future research.

In theory, a fully general correspondence between persistent singletons and inessential transitions is impossible (as witnessed by Example 3.3), which is not surprising given the difference of origins. To gauge the advantages of each of the approaches in practice, a systematic practical comparison of POR and the geometric approach is called for. We further plan extensions of the geometric approach using methods and tools from Petri nets [3,15]. This is motivated by the fact that Petri nets are closer to *both* – the geometric approach and the POR techniques – as witnessed by the study of so-called stubborn sets in Petri nets [18], which are a particular case of persistent sets [7]. The addition of Petri net inspired techniques is based on the observation that the category of future components bears similarities with the facet abstraction for occurrence nets [1] and that recent advances of the geometric approach in [4] use a notion of "weak causality", which is tightly related to classical models such as event structures and Petri net processes [19]. In the end, we expect to obtain representative experimental results, which will complement the theoretical results of the present paper.

References

[1] S. Balaguer, T. Chatain, and S. Haar. Building tight occurrence nets from reveals relations. In *ACSD*, pages 44–53. IEEE, 2011.

[2] R. Bonichon and al. Rigorous evidence of freedom from concurrency faults in industrial control software. *Computer Safety, Reliability, and Security*, pages 85–98, 2011.

[3] J.-M. Couvreur, D. Poitrenaud, and P. Weil. Branching processes of general petri nets. In *Applications and Theory of Petri Nets*, volume 6709 of *LNCS*, pages 129–148. Springer, 2011.

[4] L. Fajstrup, E. Goubault, E. Haucourt, S. Mimram, and M. Raußen. Trace spaces: An efficient new technique for state-space reduction. In *ESOP*, pages 274–294, 2012.

[5] L. Fajstrup, E. Goubault, E. Haucourt, and M. Raußen. Components of the fundamental category. *Applied Categorical Structures*, 12(1):81–108, 2004.

[6] C. Flanagan and P. Godefroid. Dynamic partial-order reduction for model checking software. In *POPL'05*, pages 110–121, 2005.

[7] P. Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem*, volume 1032 of *Lecture Notes in Computer Science*. Springer, 1996.

[8] E. Goubault. Geometry and concurrency: a user's guide. *Mathematical Structures in Computer Science*, 10(4):411–425, 2000.

[9] E. Goubault and E. Haucourt. A practical application of geometric semantics to static analysis of concurrent programs. In *Proc. of CONCUR'05*, volume 3653 of *LNCS*, pages 503–517. Springer, 2005.

[10] E. Goubault and E. Haucourt. Components of the Fundamental Category II. *Applied Categorical Structures*, 15(4):387–414, 2007.

[11] E. Goubault, E. Haucourt, and S. Krishnan. Future path-components in directed topology. In *MFPS*, volume 265 of *ENTCS*, pages 325–335, sep 2010.

[12] E. Goubault and S. Mimram. Formal relationships between geometrical and classical models for concurrency. *ENTCS*, 2012.

[13] M. Grandis. *Directed Algebraic Topology: Models of Non-Reversible Worlds*, volume 13 of *New Mathematical Monographs*. Cambridge University Press, 2009.

[14] G. Holzmann. The Model Checker SPIN. *IEEE Trans. Soft. Eng.*, 23(5):279–295, 1997.

[15] V. Khomenko and A. Mokhov. An Algorithm for Direct Construction of Complete Merged Processes. In *Applications and Theory of Petri Nets*, volume 6709 of *LNCS*, pages 89–108. Springer Berlin, 2011.

[16] S. Mimram. *Sémantique des jeux asynchrones et réécriture 2-dimensionnelle*. PhD thesis, Université Paris Diderot, UFR d'Informatique, 2008.

[17] R. Morin. Concurrent Automata vs. Asynchronous Systems. In *MFCS*, volume 3618 of *LNCS*, pages 686–698. Springer, 2005.

[18] A. Valmari. Stubborn sets for reduced state space generation. In *Applications and Theory of Petri Nets*, pages 491–515, 1989.

[19] G. Winskel. *Handbook of Logic in Computer Science*, volume 4: Semantic Modelling, chapter 1: Models for concurrency. Oxford University Press, 1995.

Coinductive Predicates and Final Sequences in a Fibration

Ichiro Hasuo Kenta Cho Toshiki Kataoka

Department of Computer Science, University of Tokyo, Japan

Bart Jacobs

ICIS, Radboud University Nijmegen, The Netherlands

Abstract

Coinductive predicates express persisting "safety" specifications of transition systems. Previous observations by Hermida and Jacobs identify coinductive predicates as suitable final coalgebras in a *fibration*—a categorical abstraction of predicate logic. In this paper we follow the spirit of a seminal work by Worrell and study final sequences in a fibration. Our main contribution is to identify some categorical "size restriction" axioms that guarantee stabilization of final sequences after ω steps. In its course we develop a relevant categorical infrastructure that relates fibrations and locally presentable categories, a combination that does not seem to be studied a lot. The genericity of our fibrational framework can be exploited for: binary relations (i.e. the logic of "binary predicates") for which a coinductive predicate is bisimilarity; constructive logics (where interests are growing in coinductive predicates); and logics for name-passing processes.

Keywords: coalgebra; (co)recursive predicate; modal logic; fibration; locally presentable category

1 Introduction

Coinductive predicates postulate properties of state-based dynamic systems that persist after a succession of transitions. In computer science, *safety properties* of nonterminating, reactive systems are examples of paramount importance. This has led to an extensive study of specification languages in the form of fixed point logics and model-checking algorithms.

In this paper we follow [26,27] (further extended in [5,20]; see also [32, Chap. 6]) and take a categorical view on coinductive predicates. Here *coalgebras* represent transition systems; a *fibration* is a "predicate logic"; and a coinductive predicate is identified as a suitable coalgebra in a fibration. Our contribution is the study of *final sequences*—an iterative construction of final coalgebras that is studied notably in [2,44]—in such a fibrational setting.

Coalgebras have been successfully used as a categorical abstraction of transition systems (see e.g. [32,41]): by varying base categories and functors, coalgebras bring general results that work for a variety of systems at once. Fixed point logics (or modal logics in general), too, have been actively studied coalgebraically: coalgebraic modal logic is a prolific research field (see [12]); their base category is typically **Sets** but works like [34] go beyond and use presheaf categories for processes in name-passing calculi; and literature including [11, 13, 43] studies coalgebraic fixed point logics.

Unlike most of these works, we follow [26, 27] and parametrize the underlying "predicate logic" too with the categorical notion of *fibration*. The conventional setting of classical logic is represented by the fibration $\begin{array}{c}\mathbf{Pred}\\ \downarrow\\ \mathbf{Sets}\end{array}$ (see Appendix A.3 for an introduction to fibrations).

However there are various other "logics" modeled as fibrations, and hence the fibrational language provides a uniform treatment of these different settings. An example is binary relations (instead of unary predicates) that form a fibration $\begin{array}{c}\mathbf{Rel}\\ \downarrow\\ \mathbf{Sets}\end{array}$ (see Appendix A.3). In this case coinductive predicates are bisimilarity (see the table, and Example 5.12 later).

fibration $\begin{array}{c}\mathbb{P}\\ \downarrow p\\ \mathbb{C}\end{array}$	$\begin{array}{c}\mathbf{Pred}\\ \downarrow\\ \mathbf{Sets}\end{array}$	$\begin{array}{c}\mathbf{Rel}\\ \downarrow\\ \mathbf{Sets}\end{array}$
coalgebra	invariant	bisimulation
final coalgebra	coindutive predicate	bisimilarity

Another example is predicates in constructive logics. They are modeled by the subobject fibration of a topos. In fact, coinductive predicates in constructive logics are an emerging research topic: coinduction is supported in the theorem prover Coq (based on the constructive *calculus of constructions*), see e.g. [6]; and, working in Coq, some interesting differences between classically equivalent (co)inductive predicates have been studied e.g. in [39].

Yet another example is modal logics for processes in various name-passing calculi. They are best modeled by the subobject fibration of a suitable (pre)sheaf category like $\mathbf{Sets}^{\mathbf{I}}$ and $\mathbf{Sets}^{\mathbf{F}}$.

1.1 Coinductive Predicates and Their Construction, Conventionally

In order to illustrate our technical contributions (§3) we first present a special case, with classical logic and Kripke models. We first introduce syntax.

Definition 1.1 (Rudimentary logic $\mathsf{R}\nu$) This fragment of the μ-calculus allows only one greatest fixed-point operator at the outermost position.

$$\mathsf{R}\nu_u \ni \alpha \ ::= \ a \mid \overline{a} \mid \Box u \mid \Diamond u \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \ ; \qquad \mathsf{R}\nu \ni \beta \ ::= \ \nu u.\, \alpha \ . \tag{1}$$

Here a belongs to the set AP of *atomic propositions*; $\overline{a}$ stands for the negation of a; and u is the only fixed-point variable (with possibly multiple occurrences).

An $\mathsf{R}\nu$-formula can be thought of as a recursive definition of a coinductive predicate. Later we will model such a "definition" categorically as a predicate lifting. A specification expressible in $\mathsf{R}\nu$ is (may-) deadlock freedom ("there is an infinite path"). It is expressed by $\nu u.\, \Diamond u$ and is our recurring example.

An $\mathsf{R}\nu$-formula is interpreted in Kripke models. Let $c = (X, \to, V)$ be a Kripke model, where X is a state space, $\to \subseteq X \times X$ is a transition relation and $V : X \to \mathcal{P}(\mathsf{AP})$ is a valuation. The conventional interpretation $[\nu u.\alpha]_c$ of $\mathsf{R}\nu$-formulas in the

Kripke model c is given as follows (see e.g. [9]). Firstly, we interpret $\alpha \in \mathsf{R}\nu_u$ as a function $[\alpha]_c : \mathcal{P}X \to \mathcal{P}X$. Concretely:

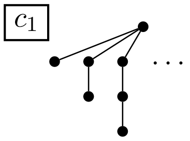
$$\begin{array}{ll} [a]_c(P) = \{x \mid a \in V(x)\} & [\overline{a}]_c(P) = \{x \mid a \not\in V(x)\} \\ [\Box u]_c(P) = \{x \mid \forall y \in X.\, (x \to y \text{ implies } y \in P)\} & [\Diamond u]_c(P) = \{x \mid \exists y \in X.\, (x \to y \text{ and } y \in P)\} \\ [\alpha \wedge \alpha']_c(P) = [\alpha]_P \cap [\alpha']_P & [\alpha \vee \alpha']_c(P) = [\alpha]_P \cup [\alpha']_P \end{array}$$

This function $[\alpha]_c$ is easily seen to be monotone, since u occurs only positively in α. Finally we define $[\nu u.\alpha]_c \subseteq X$ to be the greatest fixed point of the monotone function $[\alpha]_c : \mathcal{P}X \to \mathcal{P}X$.

The Knaster-Tarski theorem guarantees the existence of such a greatest fixed point $[\nu u.\alpha]_c$ in a complete lattice $\mathcal{P}X$. However its proof is highly nonconstructive. In contrast, a well-known construction [14] by Cousot and Cousot computes $[\nu u.\alpha]_c$ as the limit of the following descending chain (see also [9]). Here $\top$ denotes the subset $X \subseteq X$.

$$\top \;\geq\; [\alpha]_c\top \;\geq\; [\alpha]_c^2\top \;\geq\; \cdots \tag{2}$$

An issue now is the length of the chain. If $[\alpha]_c$ preserves limits $\bigwedge$ (which is the case with $\alpha \equiv \Box u$), clearly ω steps are enough and yields $\bigwedge_{i\in\omega}([\alpha]_c^i\top)$ as the greatest fixed point. This is not the case with $\alpha \equiv \Diamond u$. Indeed, for the Kripke model c_1 on the right $[\nu u.\,\Diamond u]_{c_1} \neq \bigwedge_{i\in\omega}([\Diamond u]_{c_1}^i\top)$: there is no infinite path from the root; but it satisfies $[\Diamond u]_{c_1}^i\top$ ('there is a path of length $\geq i$') for each i.



Yet the chain (2) eventually stabilizes, bounded by the size of the poset $\mathcal{P}X$. Therefore the calculation of $[\nu u.\alpha]_c$ is, in general, via *transfinite* induction. This is what we call a *state space bound* for (2).

Besides a state space bound, another (possibly better and seemingly less known) bound can be obtained from a *behavioral view*. One realizes that not only the size of the state space X but also the *branching degree* can be used to bound the length of the chain (2). For example, a result similar to [24, Thm. 2.1] states that the chain stabilizes after ω steps if the Kripke model c is *finitely branching*. This holds however large the state space X is; and also for any $\mathsf{R}\nu$-formula $\nu u.\alpha$. Notice that the model c_1 (depicted above) is not finitely branching.

1.2 Final Sequences in a Fibration

This paper is about putting the observations in §1.1 in general categorical terms. Our starting observation is that the chain (2) resembles a *final sequence*, a classic construction of a final coalgebra.

In the theory of coalgebra a *final F-coalgebra* is of prominent importance since it is a fully abstract domain with respect to the *F-behavioral equivalence*. Therefore a natural question is if a final F-coalgebra exists; the well-known Lambek lemma prohibits e.g. a final $\mathcal{P}$-coalgebra. What matters is the *size* of F: when it is suitably bounded, it is known that a final coalgebra can be constructed via the following *final F-sequence*.

$$\begin{tikzcd} 1 & F1 \arrow[l, "!"'] & \cdots \arrow[l, "F\,!"'] & F^i 1 \arrow[l, "F^{i-1}\,!"'] & \cdots \arrow[l, "F^i\,!"'] \end{tikzcd} \tag{3}$$

Here 1 is a final object in $\mathbb{C}$, and ! is the unique arrow. In particular, if F is *finitary*, a final coalgebra arises as a suitable quotient of the ω-limit of the final sequence (3). This construction in **Sets** is worked out in [44]; it is further extended to locally presentable categories (those are categories suited for speaking of "size") with additional assumptions in [2].

Turning back to coinductive predicates, indeed, the fibrational view [26,27] identifies coinductive predicates as final coalgebras in a fibration. This leads us to scrutinize final sequences in a fibration. Our main result (Thm. 3.7) is a categorical generalization of the behavioral ω-bound (§1.1)—more precisely we axiomatize categorical "size restrictions" for that bound to hold.

The conditions are formulated in the language of locally presentable categories (see e.g. [4]; also Appendix A.2); and the combination of fibrations and locally presentable categories does not seem to have been studied a lot (an exception is [37, §5.3]). We therefore develop a relevant categorical infrastructure (§5.1). Our results there include a sufficient condition for the total category $\mathrm{Sub}(\mathbb{C})$ of a subobject fibration to be locally (finitely) presentable, and the same for a family fibration $\mathrm{Fam}(\Omega)$ too. Via these results, in §5.2 we list some concrete examples of fibrations to which our results in §3 on the behavioral bounds apply. They include: $\begin{array}{c}\mathbf{Pred}\\ \downarrow\\ \mathbf{Sets}\end{array}$ (classical logic); $\begin{array}{c}\mathbf{Rel}\\ \downarrow\\ \mathbf{Sets}\end{array}$ (for bisimulation and bisimilarity); $\begin{array}{c}\mathrm{Sub}(\mathbb{C})\\ \downarrow\\ \mathbb{C}\end{array}$ for $\mathbb{C}$ that is locally finitely presentable and locally Cartesian closed (a topos is a special case); and $\begin{array}{c}\mathrm{Fam}(\Omega)\\ \downarrow\\ \mathbf{Sets}\end{array}$ for a well-founded algebraic lattice Ω.

1.3 Summary and Future Work

To summarize, our contributions are: 1) combination of the mathematical observations in [26,27] and [32, Chap. 6] for a general formulation of coinductive predicates; 2) categorical behavioral bounds for final sequences that approximate coinductive predicates; and 3) a categorical infrastructure that relates fibrations and locally presentable categories.

While our focus is on coinductive predicates, inductive ones are just as important in system verification; so are their combinations. Such mixture of induction and coinduction is studied fibrationally in [25], but over mixed inductive and coinductive data types, and not over a coalgebra. We have obtained some preliminary fibrational observations in this direction.

Search for useful coinduction proof principles is an active research topic (see e.g. [8, 28]). We are interested in the questions of whether these principles are sound in a general fibrational setting, and what novel proof principles a fibrational view can lead to.

Coalgebraic modal logic is more and more often introduced based on a Stone-like duality (see e.g. [34]). Fibrational presentation of such dualities will combine the benefits of duality-based modal logics and the current results. We are also interested in the relationship to *coalgebraic infinite traces* [10,30].

Kozen's *metric coinduction* [35] is a construction of coinductive predicates by the Banach fixed point theorem and is an alternative to the current paper's order-theoretic one. Its fibrational formulation is an interesting future topic.

Practical applications of our categorical behavioral bounds shall be pursued, too. Our results' precursor—the bounds for the final sequences in **Sets** [2,44]—have been used to bound execution of some algorithms e.g. for state minimization [3, 15, 16]. We aim at similar use. Finally, *games* are an extremely useful tool in fixed point logics (also in their coalgebraic generalization, see [11,13,43]; also [36]). We plan to investigate the use of games in the current (even more general) fibrational setting.

Organization of the Paper

In §2 we identify coinductive predicates as final coalgebras in a fibration, following the ideas of [26, 27, 32]. The main technical results are in §3, where we axiomatize size restrictions on fibrations and functors for a final sequence to stabilize after ω steps. These results are reorganized in §4 as a fibration of invariants. §5 is devoted to examples: first we develop a necessary categorical infrastructure then we discuss concrete examples.

In Appendix A we present minimal introductions to the theories of coalgebras, locally presentable categories and fibrations—the three topics that our technical developments rely on. Most proofs are deferred to Appendix B.

2 Coinductive Predicates as Final Coalgebras

In this section we follow the ideas in [26, 27, 32] and characterize coinductive predicates in various settings (for different behavior types, and in various underlying logics) in the language of fibration. An introduction to fibration is e.g. in [29]; see also Appendix A.3. In this paper for simplicity we focus on poset fibrations. It should however not be hard to move to general fibrations.

Definition 2.1 (Fibration) We refer to poset fibrations (where each fiber is a poset rather than a category) simply as *fibrations*.

$$\begin{tikzcd} \mathbb{P} \arrow[r, "\varphi"] \arrow[d, "p"'] & \mathbb{P} \arrow[d, "p"] \\ \mathbb{C} \arrow[r, "F"'] & \mathbb{C} \end{tikzcd} \qquad (4)$$

Definition 2.2 (Predicate lifting) Let $\begin{array}{c}\mathbb{P}\\ \downarrow p\\ \mathbb{C}\end{array}$ be a fibration and F be an endofunctor on $\mathbb{C}$. A *predicate lifting* of F along p is a functor $\varphi : \mathbb{P} \to \mathbb{P}$ such that (φ, F) is an endomap of fibrations. This means: that the diagram on the right commutes; and that φ preserves Cartesian arrows, that is, $\varphi(f^*Q) = (Ff)^*(\varphi Q)$. See below.

$$\begin{tikzcd} \mathbb{P} \arrow[dd, "p"] & f^*Q \arrow[r, "\overline{f}Q"] & Q & \varphi(f^*Q) \arrow[r, "\varphi(\overline{f}Q)"] \arrow[d, phantom, "\shortparallel"] & \varphi Q \\ & & & (Ff)^*(\varphi Q) \arrow[ru, bend right, "\overline{Ff}(\varphi Q)"'] & \\ \mathbb{C} & X \arrow[r, "f"] & Y & FX \arrow[r, "Ff"] & FY \end{tikzcd} \qquad (5)$$

In the prototype example $\begin{array}{c}\mathbf{Pred}\\ \downarrow\\ \mathbf{Sets}\end{array}$, the above definition coincides (see [32]) with the one used in coalgebraic modal logic (see e.g. [12])—presented as a (monotone) natural transformation $2^{(\_)} \overset{\varphi}{\Rightarrow} 2^{F(\_)} : \mathbf{Sets}^{\mathrm{op}} \to \mathbf{Sets}$.

We think of predicate liftings as (co)recursive definitions of coinductive predicates (see Example 2.4). On top of it, we identify coinductive predicates (and invariants) as coalgebras in a fiber.

Definition 2.3 (Invariant, coinductive predicate) Let φ be a predicate lifting of F along $\begin{smallmatrix}\mathbb{P}\\ \downarrow p\\ \mathbb{C}\end{smallmatrix}$; and $X \xrightarrow{c} FX$ be a coalgebra in $\mathbb{C}$. They together induce an endofunctor (a monotone function) on the fiber $\mathbb{P}_X$, namely $\mathbb{P}_X \xrightarrow{\varphi} \mathbb{P}_{FX} \xrightarrow{c^*} \mathbb{P}_X$, where φ restricts to $\mathbb{P}_X \to \mathbb{P}_{FX}$ because of (4).

(i) A *φ-invariant* in c is a $(c^* \circ \varphi)$-coalgebra in $\mathbb{P}_X$, that is, an object $P \in \mathbb{P}_X$ such that $P \leq c^*(\varphi P)$ in $\mathbb{P}_X$.

(ii) The *φ-coinductive predicate* in c is the final $(c^* \circ \varphi)$-coalgebra (if it exists). Its carrier shall be denoted by $[\![\nu\varphi]\!]_c$. It is therefore the largest φ-invariant in c; Lambek's lemma yields that $[\![\nu\varphi]\!]_c = (c^* \circ \varphi)([\![\nu\varphi]\!]_c)$.

Example 2.4 ($\mathsf{R}\nu$) The conventional interpretation $[\nu u.\alpha]_c$ (described in §1.1) of $\mathsf{R}\nu$-formulas is a special case of Def. 2.3. Indeed, let us work in the fibration $\begin{smallmatrix}\mathbf{Pred}\\ \downarrow\\ \mathbf{Sets}\end{smallmatrix}$, and with the endofunctor $F_{\mathrm{K}} = \mathcal{P}(\mathsf{AP}) \times \mathcal{P}(\_)$ on **Sets**. An F_{K}-coalgebra $X \xrightarrow{c} \mathcal{P}(\mathsf{AP}) \times \mathcal{P}X$ is precisely a Kripke model: c combines a valuation $X \to \mathcal{P}(\mathsf{AP})$ and the map $X \to \mathcal{P}X$ that carries a state to the set of its successors. To each formula $\alpha \in \mathsf{R}\nu_u$ we associate a predicate lifting φ_α of F_{K}. This is done inductively as follows.

$$\begin{array}{ll}
\varphi_a(U \subseteq X) = \bigl(\{V \in F_{\mathrm{K}}X \mid a \in \pi_1(V)\} \subseteq F_{\mathrm{K}}X\bigr) & \varphi_{\overline{a}}(U \subseteq X) = \bigl(\{V \mid a \notin \pi_1(V)\} \subseteq F_{\mathrm{K}}X\bigr) \\
\varphi_{\Box u}(U \subseteq X) = \bigl(\{V \mid \pi_2(V) \subseteq U\} \subseteq F_{\mathrm{K}}X\bigr) & \varphi_{\Diamond u}(U \subseteq X) = \bigl(\{V \mid \exists x \in U.\, x \in \pi_2(V)\} \subseteq F_{\mathrm{K}}X\bigr) \\
\varphi_{\alpha\wedge\alpha'}(U \subseteq X) = \bigl((\varphi_\alpha U \cap \varphi_{\alpha'} U) \subseteq F_{\mathrm{K}}X\bigr) & \varphi_{\alpha\vee\alpha'}(U \subseteq X) = \bigl((\varphi_\alpha U \cup \varphi_{\alpha'} U) \subseteq F_{\mathrm{K}}X\bigr)
\end{array} \tag{6}$$

In the above, π_1 and π_2 denote the projections from $F_{\mathrm{K}}X = \mathcal{P}(\mathsf{AP}) \times \mathcal{P}X$. Then it is easily seen by induction that $[\![\nu\varphi_\alpha]\!]_c$ in Def. 2.3 coincides with the conventional interpretation $[\nu u.\alpha]_c$ described in §1.1.

In fact, the predicate liftings φ_α in (6) are the ones commonly used in coalgebraic modal logic (where they are presented as natural transformations). We point out that the same definition of φ_α—they are written in the internal language of toposes—works for the subobject fibration $\begin{smallmatrix}\mathrm{Sub}(\mathbb{C})\\ \downarrow\\ \mathbb{C}\end{smallmatrix}$ of any topos $\mathbb{C}$. Therefore the categorical definition of coinductive predicates (Def. 2.3) allows us to interpret the language $\mathsf{R}\nu$ in constructive underlying logics. Suitable completeness of $\mathbb{C}$ ensures that a final $(c^* \circ \varphi)$-coalgebra in Def. 2.3 exists.

Proposition 2.5 *Let φ be a predicate lifting of F along $\begin{smallmatrix}\mathbb{P}\\ \downarrow p\\ \mathbb{C}\end{smallmatrix}$; $X \xrightarrow{c} FX$ be a coalgebra in $\mathbb{C}$; and $P \in \mathbb{P}_X$. We have $P \leq [\![\nu\varphi]\!]_c$ if and only if there exists a φ-invariant Q such that $P \leq Q$.* □

The proposition is trivial but potentially useful. It says that an invariant can be used as a "witness" for a coinductive predicate. This is how bisimilarity is commonly established; and it can be used e.g. in [1, §6] as an alternative to the metric coinduction principle used there.[1]

[1] To be precise: only if we take PE in [1] as an atomic proposition (and that is essentially what is done in the proofs in [1, §6]). Our future work on nested μ's and ν's will more adequately address the situation.

Remark 2.6 The coalgebraic modal logic literature exploits the fact that there can be many predicate liftings (in the form of natural transformations) of the same functor F. Different predicate liftings correspond to different modalities (such as $\Box$ vs. $\Diamond$ for the same functor $\mathcal{P}$). This view of predicate liftings is also the current paper's (see Example 2.4).

In contrast, in fibrational studies like [5, 20, 26, 27], use of predicate liftings has focused on the validity of the *(co)induction proof principle*. For such purposes it is necessary to choose a predicate lifting φ that is "comprehensive enough," covering all the possible F-behaviors. In fact, it is common in these studies that "the" predicate lifting, denoted by $\mathrm{Pred}(F)$, is assigned to a functor F. An exception is [31].

3 Final Sequences in a Fibration

Here we present our main technical result (Thm. 3.7). It generalizes known behavioral ω-bounds (like [24, Thm. 2.1]; see §1.1); and claims that the chain (2) for a coinductive predicate stabilizes after ω steps, assuming that the behavior type functor F and the underlying logic $\begin{array}{c}\mathbb{P}\\ \downarrow p\\ \mathbb{C}\end{array}$ are "finitary" in a suitable sense (but no size restriction on φ).

3.1 Size Restrictions on a Fibration

We axiomatize finitariness conditions in the language of locally presentable categories (see Appendix A.2 for a minimal introduction). Singling out these conditions lies at the heart of our technical contribution.

Definition 3.1 (LFP category) A category $\mathbb{C}$ is *locally finitely presentable (LFP)* if it is cocomplete and it has a (small) set $\mathbb{F}$ of finitely presentable (FP) objects such that every object is a directed colimit of objects in $\mathbb{F}$.

Definition 3.2 (Finitely determined fibration) A (poset) fibration $\begin{array}{c}\mathbb{P}\\ \downarrow p\\ \mathbb{C}\end{array}$ is *finitely determined* if it satisfies the following.

(i) $\mathbb{C}$ is LFP, with a set $\mathbb{F}$ of FP objects (as in Def. 3.1).

(ii) $\begin{array}{c}\mathbb{P}\\ \downarrow p\\ \mathbb{C}\end{array}$ has fiberwise limits and colimits.

(iii) For arbitrary $X \in \mathbb{C}$, let $(X_I)_{I \in \mathbb{I}}$ be the canonical diagram for X with respect to $\mathbb{F}$ (i.e. $\mathbb{I} = (\mathbb{F} \downarrow X)$), with a colimiting cocone $(X_I \overset{\kappa_I}{\to} X)_{I \in \mathbb{I}}$. Then for any $P, Q \in \mathbb{P}_X$,

$$P \leq Q \quad \Longleftrightarrow \quad \kappa_I^* P \leq \kappa_I^* Q \text{ in } \mathbb{P}_{X_I} \text{ for each } I \in \mathbb{I}.$$

The intuition of Cond. iii) is that a predicate $P \in \mathbb{P}_X$ (over arbitrary $X \in \mathbb{C}$) is determined by its restrictions $(\kappa_I^* P)_{I \in \mathbb{I}}$ to FP objects X_I. One convenient sufficient condition for Cond. iii) is that the total category $\mathbb{P}$ is itself LFP, with its FP objects above the FP objects in $\mathbb{C}$ (Cor. 5.3). We note that Cond. i) guarantees, since LFP implies completeness, an (ω^{op}-)limit $F^\omega 1$ of the final F-sequence (3). However this does not mean (nor we need for later) that $F^\omega 1$ carries a final F-coalgebra (it fails for $F = \mathcal{P}_\omega$; see [44]).

Definition 3.3 (Well-founded fibration) A *well-founded fibration* is a finitely determined fibration that further satisfies:

(iv) If $X \in \mathbb{F}$ (hence FP), the fiber $\mathbb{P}_X$ is such that: the category $\mathbb{P}_X^{\mathrm{op}}$ consists solely of FP objects.

Since $\mathbb{P}_X$ is complete, this is equivalent to: there is no (ω^{op}-)chain $P_0 > P_1 > \cdots$ in $\mathbb{P}_X$ that is strictly descending.

We note that the following stronger variant of the condition

(iv') For *any* $X \in \mathbb{C}$, there is no strictly descending ω^{op}-chain in $\mathbb{P}_X$

rarely holds (it fails in $\begin{smallmatrix}\mathbf{Pred}\\ \downarrow\\ \mathbf{Sets}\end{smallmatrix}$). The original Cond. iv) holds in many examples (as we will see later in §5) thanks to the restriction that X is FP.

The following trivial fact is written down for the record.

Lemma 3.4 *A finitely determined fibration* $\begin{smallmatrix}\mathbb{P}\\ \downarrow p\\ \mathbb{C}\end{smallmatrix}$ *is well-founded if* $\mathbb{P}_X$ *is a finite category for each* $X \in \mathbb{F}$. □

3.2 Final Sequences in a Fibration

The following result from [29, Prop. 9.2.1] is crucial in our development.

Lemma 3.5 *Let* $\begin{smallmatrix}\mathbb{P}\\ \downarrow p\\ \mathbb{C}\end{smallmatrix}$ *be a fibration, with* $\mathbb{C}$ *being complete. Then* p *has fiberwise limits if and only if* $\mathbb{P}$ *is complete and* $p : \mathbb{P} \to \mathbb{C}$ *preserves limits. If this is the case, a limit of a small diagram* $(P_I)_{I \in \mathbb{I}}$ *in* $\mathbb{P}$ *can be given by*

$$\textstyle\bigwedge_{I \in \mathbb{I}} (\pi_I^* P_I) \qquad \textit{over } \mathrm{Lim}_{I \in \mathbb{I}} X_I .$$

Here $X_I := pP_I$*;* $(\mathrm{Lim}_{I \in \mathbb{I}} X_I \overset{\pi_I}{\to} X_I)_{I \in \mathbb{I}}$ *is a limiting cone in* $\mathbb{C}$*; and* $\bigwedge_{I \in \mathbb{I}}$ *denotes the limit in the fiber* $\mathbb{P}_{\mathrm{Lim}_I X_I}$. □

Fig. 1 presents two sequences. Here we assume that $\begin{smallmatrix}\mathbb{P}\\ \downarrow p\\ \mathbb{C}\end{smallmatrix}$ is finitely determined (Def. 3.2) and that φ is a predicate lifting of F. In the bottom diagram (in $\mathbb{C}$), the

$$\begin{tikzcd}
\boxed{\mathbb{P}} & & & & & \varphi^\omega \top_1 \arrow[dlllll, bend right] \arrow[dllll, bend right] \arrow[dll, bend right] \\
\top_1 & \varphi \top_1 \arrow[l] & \cdots \arrow[l] & \varphi^i \top_1 \arrow[l] & \cdots \arrow[l] & \\
& & & & & \varphi^{\omega+1} \top_1 \arrow[ulllll, bend left] \arrow[ullll, bend left] \arrow[ull, bend left] \arrow[uu, dashed, bend right, "b'"']
\end{tikzcd}$$

$$\begin{tikzcd}
\boxed{\mathbb{C}} & & & & & F^\omega 1 \arrow[dlllll, bend right] \arrow[dllll, bend right] \arrow[dll, bend right, "\pi_i"] \\
1 & F1 \arrow[l, "!"] & \cdots \arrow[l] & F^i 1 \arrow[l, "F^{i-1} !"'] & \cdots \arrow[l, "F^i !"'] & \\
& & & & & F^{\omega+1} 1 \arrow[ulllll, bend left] \arrow[ullll, bend left] \arrow[ull, bend left, "F\pi_{i-1}"'] \arrow[uu, dashed, bend right, "b"']
\end{tikzcd}$$

Figure 1. Final sequences in a fibration

object $1 \in \mathbb{C}$ is a final one (it exists since LFP implies completeness); $F1 \xrightarrow{!} 1$ is the unique map; $F^{\omega+1}1 := F(F^{\omega}1)$; and b is a unique mediating arrow to the limit $F^{\omega}1$. In the top diagram (in $\mathbb{P}$), the object $\top_1$ is the final object in the fiber $\mathbb{P}_1$; by Lem. 3.5 this is precisely a final object in the total category $\mathbb{P}$. Hence this diagram is nothing but a final sequence for the functor φ in $\mathbb{P}$. A limit $\varphi^{\omega}\top_1$ of this final sequence exists, again by Lem. 3.5, and moreover it can be chosen above $F^{\omega}1$. We define $\varphi^{\omega+1}\top_1 := \varphi(\varphi^{\omega}\top_1)$.

Lemma 3.6 (Key lemma) *Let* $\begin{array}{c}\mathbb{P}\\ \downarrow p\\ \mathbb{C}\end{array}$ *be a well-founded fibration;* $F : \mathbb{C} \to \mathbb{C}$ *be finitary; and* φ *be a predicate lifting of* F*. Then the final* φ*-sequence stabilizes after* ω *steps. More precisely: in Fig. 1, we have* $\varphi^{\omega+1}\top_1 = b^*(\varphi^{\omega}\top_1)$*.*

The object $\varphi^{\omega}\top_1$ is a "prototype" of φ-coinductive predicates in various coalgebras. This is one content of the following main theorem.

It is standard that a coalgebra $X \xrightarrow{c} FX$ in $\mathbb{C}$ induces a cone over the final F-sequence, and hence a mediating arrow $X \to F^{\omega}1$ (see below). Concretely, $c_i : X \to F^i 1$ is defined inductively by: $X \xrightarrow{c_0} 1$ is $!$; and c_{i+1} is the composite $X \xrightarrow{c} FX \xrightarrow{Fc_i} F^{i+1}1$. The induced arrow to the limit $F^{\omega}1$ is denoted by c_{ω}.

$$\begin{tikzcd}
 & & & & F^{\omega}1 \arrow[llll, bend right] \arrow[lll, bend right] \arrow[ld, bend right, "\pi_i"'] \\
1 & F1 \arrow[l, "!"] & \cdots \arrow[l] & F^i 1 \arrow[l] & \cdots \arrow[l] \\
 & & & & X \arrow[llll, bend left] \arrow[lll, bend left] \arrow[lu, bend left, "c_i"] \arrow[uu, dashed, bend right, "c_{\omega}"']
\end{tikzcd} \tag{7}$$

Theorem 3.7 (Main result) *Let* $\begin{array}{c}\mathbb{P}\\ \downarrow p\\ \mathbb{C}\end{array}$ *be a well-founded fibration;* $F : \mathbb{C} \to \mathbb{C}$ *be a finitary functor;* φ *be a predicate lifting of* F *along* p*; and* $X \xrightarrow{c} FX$ *be a coalgebra in* $\mathbb{C}$*.*

(i) *The* φ*-coinductive predicate* $[\![\nu\varphi]\!]_c$ *in* c *(Def. 2.3) exists. It is obtained by the following reindexing of* $\varphi^{\omega}\top_1$*, where* c_{ω} *is the mediating map in (7).*

$$[\![\nu\varphi]\!]_c \;=\; c_{\omega}^*(\varphi^{\omega}\top_1) \tag{8}$$

(ii) *Moreover, the predicate* $[\![\nu\varphi]\!]_c$ *is the limit of the following* ω^{op}*-chain in the fiber* $\mathbb{P}_X$

$$\top_X \;\geq\; (c^* \circ \varphi)(\top_X) \;\geq\; (c^* \circ \varphi)^2(\top_X) \;\geq\; \cdots \;, \tag{9}$$

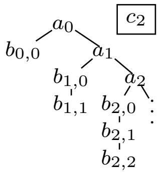
that stabilizes after ω *steps. That is,* $[\![\nu\varphi]\!]_c = \bigwedge_{i\in\omega}(c^* \circ \varphi)^i(\top_X)$*.* □

Example 3.8 ($\mathsf{R}\nu$) We continue Example 2.4 and derive from Thm. 3.7 the behavioral bound result described in §1.1: the chain (2) stabilizes after ω steps, for each $\alpha \in \mathsf{R}\nu_u$ and each *finitely branching* Kripke model c.

Indeed, the latter is the same thing as a coalgebra $X \xrightarrow{c} F_{\mathrm{fbK}}X$, where $F_{\mathrm{fbK}} = \mathcal{P}(\mathsf{AP}) \times \mathcal{P}_{\omega}(\_)$. Compared to F_{K} in Example 2.4 the powerset functor is restricted from $\mathcal{P}$ to $\mathcal{P}_{\omega}$; this makes F_{fbK} a finitary functor. Still the same definition of φ_{α} defines a predicate lifting of F_{fbK}. Thm. 3.7.ii can then be applied to the fibration $\begin{array}{c}\mathbf{Pred}\\ \downarrow\\ \mathbf{Sets}\end{array}$ (easily seen to be well-founded, Example 5.11), the finitary functor F_{fbK} and the predicate lifting φ_{α} for each α. It is not hard to see that the function $[\alpha]_c : \mathcal{P}X \to \mathcal{P}X$ in §1.1 coincides with $c^* \circ \varphi_{\alpha} : \mathbf{Pred}_X \to \mathbf{Pred}_X$ (note that

$\mathbf{Pred}_X \cong 2^X \cong \mathcal{P}X$); thus the chain (2) coincides with (9) that stabilizes after ω steps by Thm. 3.7.

Remark 3.9 The ω-bound of the length of the chain (9) is sharp.



A (counter)example is given in the setting of Example 3.8, by the predicate lifting $\varphi_{\Diamond u}$ and the coalgebra (i.e. Kripke structure) c_2 on the right. There $b_{i,i}$ has no successors. Indeed, while $[\![\nu\varphi_{\Diamond u}]\!]_{c_2}$ is $\{a_i \mid i \in \omega\}$, its i-th approximant $((c_2)^*_i \circ \varphi^i_{\Diamond u})(\top_X)$ in (9) contains $b_{i,0}$ too.

Remark 3.10 It is notable that Thm. 3.7 imposes no size restrictions on $\varphi : \mathbb{P} \to \mathbb{P}$. Being a predicate lifting is enough.

Final F-sequences are commonly used for the construction of a final F-coalgebra. It is not always the case, however, that the limit $F^\omega 1$ is itself the carrier of a final coalgebra (even for finitary F; see [44, §5]). One obtains a final coalgebra either by: 1) quotienting $F^\omega 1$ by the behavioral equivalence (see e.g. [40]); or 2) continuing the final sequence till $\omega + \omega$ steps. The latter construction is worked out in [44] (in **Sets**) and in [2] (in LFP $\mathbb{C}$ with additional assumptions). Its relevance to the current work is yet to be investigated.

Coalgebra morphisms are compatible with coinductive predicates. This fact, like Prop. 2.5, is potentially useful in establishing coinductive predicates.

Proposition 3.11 *Let $f : X \to Y$ be a coalgebra morphism from $X \xrightarrow{c} FY$ to $Y \xrightarrow{d} FY$. In the setting of Lem. 3.6 and Thm. 3.7:*

(i) *If $Q \in \mathbb{P}_Y$ is a φ-invariant in d, so is $f^*Q \in \mathbb{P}_X$ in c.*

(ii) *We have $[\![\nu\varphi]\!]_c = f^*\big([\![\nu\varphi]\!]_d\big)$.* □

Remark 3.12 The current paper focuses on *finitely presentable* objects, *finitary* functors, etc.—i.e. the ω-presentable setting (see [4, §1.B]). This is for the simplicity of presentation: the results, as usual (as e.g. in [34]), can be easily generalized to the λ-presentable setting for an arbitrary regular cardinal λ. In such an extended setting we obtain a behavioral λ-bound.

4 A Fibration of Invariants

We organize the above observations in a more abstract fibered setting. The technical results are mostly standard; see e.g. [26, 27] and [32, Chap.6].

We write $\mathbf{Coalg}(F)$ for the category of F-coalgebras.

Proposition 4.1 *Let φ be a predicate lifting of F along $\begin{array}{c}\mathbb{P}\\ \downarrow p\\ \mathbb{C}\end{array}$. Then the fibration $\begin{array}{c}\mathbb{P}\\ \downarrow p\\ \mathbb{C}\end{array}$ is lifted to a fibration $\begin{array}{c}\mathbf{Coalg}(\varphi)\\ \downarrow \overline{p}\\ \mathbf{Coalg}(F)\end{array}$, with two forgetful functors forming a map of fibrations from the latter to the former.* □

The next observation explains the current section's title.

Proposition 4.2 *Let* $\begin{array}{c}\mathbf{Coalg}(\varphi)\\ \downarrow\overline{p}\\ \mathbf{Coalg}(F)\end{array}$ *be the lifted fibration in Prop. 4.1. For each coalgebra* $X \overset{c}{\to} FX$*, the fiber over* c *coincides with the poset of* φ*-invariants in* c*. That is:*

$$\begin{tikzcd}[column sep=small] \mathbf{Coalg}(\varphi)_{X \overset{c}{\to} FX} \arrow[rr, "\cong"] \arrow[dr] & & \mathbf{Coalg}(c^{*} \circ \varphi) \arrow[dl] \\ & \mathbb{P}_{X} & \end{tikzcd}.$$

□

Therefore Thm. 3.7.i) and Prop. 3.11.ii) state the fibration $\begin{array}{c}\mathbf{Coalg}(\varphi)\\ \downarrow\overline{p}\\ \mathbf{Coalg}(F)\end{array}$ has fiberwise final objects. (At least part of) this statement itself is shown quite easily using the Knaster-Tarski theorem (each fiber is a complete lattice). Our contribution is its concrete construction as an ω^{op}-limit (Thm. 3.7.ii).

The following is an immediate consequence of Lem. 3.5.

Corollary 4.3 *Let* φ *be a predicate lifting of* F *along* $\begin{array}{c}\mathbb{P}\\ \downarrow p\\ \mathbb{C}\end{array}$*; and assume that a final* F*-coalgebra exists. The following are equivalent.*

(i) *The coinductive predicate* $[\![\nu\varphi]\!]_{c}$ *exists for each coalgebra* $c : X \to FX$*. Moreover they are preserved by reindexing (along coalgebra morphisms).*

(ii) *There exists a final* φ*-coalgebra that is above a final* F*-coalgebra.* □

5 Examples of Fibrations

5.1 Examples at Large

Here are several results that ensure a fibration to be finitely determined or well-founded, and hence enable us to apply Thm. 3.7. Some of them are well-known; others—especially those which relate fibrations and locally (finitely) presentable categories, including Lem. 5.4 and Cor. 5.7—seem to be new.

Lemma 5.1 [29, Prop. 5.4.7] *An (elementary) topos is a locally Cartesian closed category (LCCC).* □

The following results provide sufficient conditions for a fibration to be finitely determined (Def. 3.2). Recall that a full subcategory $\mathbb{F}$ of $\mathbb{P}$ is said to be *dense* if each object $P \in \mathbb{P}$ is a colimit of a diagram in $\mathbb{F}$.

Lemma 5.2 *Let* $\begin{array}{c}\mathbb{P}\\ \downarrow p\\ \mathbb{C}\end{array}$ *be a fibration with fiberwise limits and colimits. Assume further that* $\mathbb{C}$ *is LFP with a set* $\mathbb{F}_{\mathbb{C}}$ *of FP objects (as in Def. 3.1). If the total category* $\mathbb{P}$ *has a dense subcategory* $\mathbb{F}_{\mathbb{P}}$ *such that every* $R \in \mathbb{F}_{\mathbb{P}}$ *is above* $\mathbb{F}_{\mathbb{C}}$ *(i.e.* $pR \in \mathbb{F}_{\mathbb{C}}$*), then* p *is finitely determined.* □

Corollary 5.3 *Let* $\begin{array}{c}\mathbb{P}\\ \downarrow p\\ \mathbb{C}\end{array}$ *be a fibration with fiberwise limits and colimits, where* $\mathbb{C}$ *is LFP with a set* $\mathbb{F}_{\mathbb{C}}$ *of FP objects (in Def. 3.1). If the total category* $\mathbb{P}$ *is also LFP, with a set* $\mathbb{F}_{\mathbb{P}}$ *of FP objects (as in Def. 3.1) chosen so that every* $R \in \mathbb{F}_{\mathbb{P}}$ *is above* $\mathbb{F}_{\mathbb{C}}$*, then* p *is finitely determined.* □

The following is one of the results that are less trivial.

Lemma 5.4 *Let $\mathbb{C}$ be an LFP category with $\mathbb{F}$ being a set of FP objects (as in Def. 3.1). Assume that $\mathbb{C}$ is at the same time an LCCC. Then the total category* $\mathrm{Sub}(\mathbb{C})$ *of the subobject fibration is LFP: the set* $\mathbb{F}_{\mathrm{Sub}(\mathbb{C})} := \{\, (P \rightarrowtail X) \mid P, X \in \mathbb{F} \}$ *consists of FP objects in* $\mathrm{Sub}(\mathbb{C})$*; and every object* $(Q \rightarrowtail Y) \in \mathrm{Sub}(\mathbb{C})$ *is a colimit of a directed diagram in* $\mathbb{F}_{\mathrm{Sub}(\mathbb{C})}$. □

It follows from Lem. 5.1, 5.4, and Cor. 5.3 that the internal logic of a topos that is LFP is finitely determined.

Corollary 5.5 *Let $\mathbb{C}$ be LFP and at the same time a topos (or more generally an LCCC). Then the subobject fibration* $\begin{smallmatrix}\mathrm{Sub}(\mathbb{C})\\ \downarrow\\ \mathbb{C}\end{smallmatrix}$ *is finitely determined.* □

We turn to the family fibration $\begin{smallmatrix}\mathrm{Fam}(\Omega)\\ \downarrow\\ \mathbf{Sets}\end{smallmatrix}$ over a poset Ω (see Appendix A.3).

Lemma 5.6 *Let Ω be an* algebraic lattice*, i.e. a complete lattice in which each element is a join of compact elements. (Equivalently, Ω is LFP when considered as a category.) Then the set*

$$\mathbb{F}_{\mathrm{Fam}(\Omega)} := \{\, f : X \to \Omega \mid X \textit{ is finite; for each } x \in X,\ f(x) \textit{ is compact in } \Omega \,\} \tag{10}$$

consists of finitely generated objects and is dense in $\mathrm{Fam}(\Omega)$*. Therefore by Lem. 5.2,* $\begin{smallmatrix}\mathrm{Fam}(\Omega)\\ \downarrow\\ \mathbf{Sets}\end{smallmatrix}$ *is finitely determined.* □

It is known that the existence of a dense set of FG objects (like $\mathbb{F}_{\mathrm{Fam}(\Omega)}$ in Lem. 5.6) ensures the category to be locally λ-presentable. This is however for some regular cardinal λ that is possibly bigger than ω. See [4, Thm. 1.70].

Corollary 5.7 *Let Ω be an algebraic lattice. Then the total category* $\mathrm{Fam}(\Omega)$ *of* $\begin{smallmatrix}\mathrm{Fam}(\Omega)\\ \downarrow\\ \mathbf{Sets}\end{smallmatrix}$ *is locally presentable.* □

We turn to the notion of well-founded fibration (Def. 3.3; see also Lem. 3.4).

Example 5.8 (Presheaf categories) Let $\mathbb{A}$ be small. The presheaf category $\mathbf{Sets}^{\mathbb{A}}$ is LFP: the set $\mathbb{F}$ of finite colimits of representable presheaves $\mathbf{y}A$, where $\mathbf{y}A = \mathbb{A}(A, \_)$, satisfies the conditions of Def. 3.1.

The coming results are less trivial, too.

Lemma 5.9 *Let $\mathbb{A}$ be small. For any $X \in \mathbb{A}$,* $\mathrm{Sub}(\mathbf{y}X)$ *is finite if and only if the subset* $\{\mathrm{Im}(\mathbf{y}A \xrightarrow{\mathbf{y}f} \mathbf{y}X) \mid A \in \mathbb{A}, f\colon X \to A\} \subseteq \mathrm{Sub}(\mathbf{y}X)$ *is finite.*

As a special case, if every arrow f with domain $X \in \mathbb{A}$ factors $f = m \circ e$ as a split mono m followed by an epi e, then $\mathrm{Sub}(\mathbf{y}X)$ *is finite if and only if* $\mathrm{Quot}(X)$ *is finite. Here* $\mathrm{Quot}(X)$ *denotes the set of quotient objects of X.* □

Corollary 5.10 *If one of the conditions in Lem. 5.9 holds, the fibration* $\begin{smallmatrix}\mathrm{Sub}(\mathbf{Sets}^{\mathbb{A}})\\ \downarrow\\ \mathbf{Sets}^{\mathbb{A}}\end{smallmatrix}$ *is well-founded.* □

5.2 Concrete Examples

Example 5.11 (Pred) The fibration $\begin{array}{c}\mathbf{Pred}\\ \downarrow\\ \mathbf{Sets}\end{array}$ for the conventional setting of classical logic is easily seen to be well-founded. In particular, $\mathbf{Pred}_X \cong \mathcal{P}X$ is finite if X is FP (i.e. finite). Therefore to any finitary F and any predicate lifting φ, the results in §3 apply.

The (interpretations of the) formulas in $\mathsf{R}\nu$ (see Example 3.8) are examples of coinductive predicates in $\begin{array}{c}\mathbf{Pred}\\ \downarrow\\ \mathbf{Sets}\end{array}$. Besides them, the study of coalgebraic modal logic has identified many predicate liftings for many functors F (probabilistic systems, neighborhood frames, strategy frames, weighted systems, etc.; see e.g. [12] and the references therein). These "modalities" all define coinductive predicates, to which the results in §3 may apply.

Example 5.12 (Rel) The fibration $\begin{array}{c}\mathbf{Rel}\\ \downarrow\\ \mathbf{Sets}\end{array}$ can be introduced from $\begin{array}{c}\mathbf{Pred}\\ \downarrow\\ \mathbf{Sets}\end{array}$ via change-of-base; concretely, an object of $\mathbf{Rel}$ is a pair (X, R) of a set X and a relation $R \subseteq X \times X$; an arrow $f : (X, R) \to (Y, S)$ is a function $f : X \to Y$ such that xRx' implies $f(x)Sf(x')$. See [29, p. 14].

This fibration is also easily seen to be well-founded; therefore to any finitary F the results in §3 apply. A predicate lifting φ along $\begin{array}{c}\mathbf{Rel}\\ \downarrow\\ \mathbf{Sets}\end{array}$ is more commonly called a *relation lifting* [27]; by choosing a suitable φ (a "sufficiently comprehensive" one) like in [27], a φ-invariant is precisely a bisimulation relation, and the φ-coinductive predicate is bisimilarity. We expect that the ω-behavioral bound in Thm. 3.7 can be used to bound execution of bisimilarity checking algorithms by partition refinement (for many different functors F).

In the following example, one can think of Ω as a Heyting algebra, and then the underlying logic becomes constructive.

Example 5.13 ($\mathrm{Fam}(\Omega)$) Let Ω be an algebraic lattice that has no strictly descending (ω^{op}-)chains. Then the family fibration $\begin{array}{c}\mathrm{Fam}(\Omega)\\ \downarrow\\ \mathbf{Sets}\end{array}$ is well-founded (see Lem. 5.6). Therefore to any finitary F the results in §3 apply. It is not hard to interpret the language $\mathsf{R}\nu$ in this setting, by defining predicate liftings similar to (6). This gives examples of coinductive predicates in $\begin{array}{c}\mathrm{Fam}(\Omega)\\ \downarrow\\ \mathbf{Sets}\end{array}$.

Presheaf Examples

Let $\mathbf{F}$ be the category of natural numbers as finite sets (i.e. $n = \{0, 1, \ldots, n-1\}$) and all functions between them; $\mathbf{F}_+$ be its full subcategory of nonzero natural numbers; and $\mathbf{I}$ be the category of natural numbers and injective functions. Coalgebras in the presheaf categories $\mathbf{Sets}^{\mathbf{F}}$, $\mathbf{Sets}^{\mathbf{F}_+}$ and $\mathbf{Sets}^{\mathbf{I}}$ are commonly used for modeling processes in various name-passing calculi. For the π-calculus $\mathbf{Sets}^{\mathbf{I}}$ has been found appropriate (see e.g. [17,18]); while for the fusion calculus we do need non-injective functions in $\mathbf{F}$ or $\mathbf{F}_+$ (see [38,42]).

Inspired by [34], we are interested in coinductive predicates for such processes. They are naturally modeled in the subobject fibration of a presheaf category. Here we find a distinction: the subobject fibrations of $\mathbf{Sets}^{\mathbf{F}}$ and $\mathbf{Sets}^{\mathbf{F}_+}$ are well-founded; but that of $\mathbf{Sets}^{\mathbf{I}}$ is not. In view of Cor. 5.5 and Example 5.8, the only condition to check is Cond. iv) in Def. 3.3.

Example 5.14 **(**$\mathrm{Sub}(\mathbf{Sets}^{\mathbf{F}})$**,** $\mathrm{Sub}(\mathbf{Sets}^{\mathbf{F}_+})$**)** The subobject fibration $\begin{array}{c}\mathrm{Sub}(\mathbf{Sets}^{\mathbf{F}_+})\\ \downarrow\\ \mathbf{Sets}^{\mathbf{F}_+}\end{array}$ is well-founded: this is shown by Cor. 5.10. An important fact here is that in **Sets** a mono with a nonempty domain splits.

The subobject fibration $\begin{array}{c}\mathrm{Sub}(\mathbf{Sets}^{\mathbf{F}})\\ \downarrow\\ \mathbf{Sets}^{\mathbf{F}}\end{array}$ is well-founded, too. To show that $\mathrm{Sub}(\mathbf{y}0)$ is finite, we appeal to the first half of Lem. 5.9: we observe that the set $\{\mathrm{Im}\,\mathbf{y}f \mid n \in \mathbf{F}, f\colon 0 \to n\}$ is equal to the two-element set $\big\{\,\mathrm{Im}(\mathbf{y}(0 \overset{\mathrm{id}_0}{\to} 0)), \mathrm{Im}(\mathbf{y}(0 \overset{!}{\to} 1))\,\big\}$ since $0 \overset{!}{\to} n$ and $0 \overset{!}{\to} m$ factor through each other, for each $n, m \geq 1$.

We turn to functors F and φ. In modeling processes of name-passing calculi as coalgebras in these categories, one typically uses endofunctors F that are constructed from the following building blocks. Let $\mathbf{N} \in \{\mathbf{F}, \mathbf{F}_+, \mathbf{I}\}$.

- Constant functors, binary sum $+$, binary product $\times$, and exponentials $(\_)^X$. These are much like for polynomial functors on **Sets**. An important example of the first is the *name* presheaf $\mathcal{N} = \mathrm{Hom}(1, \_) \in \mathbf{Sets}^{\mathbf{N}}$.
- The *abstraction* functor $\delta : \mathbf{Sets}^{\mathbf{N}} \to \mathbf{Sets}^{\mathbf{N}}$ given by $\delta X = X(\_ + 1)$.
- The free semilattice functor $\mathcal{P}_{\mathrm{f}}$ for finite branching. This captures Kuratowski finiteness and suitable in $\mathbf{Sets}^{\mathbf{I}}$. See e.g. [17, 42].
- In $\mathbf{Sets}^{\mathbf{F}}$ and $\mathbf{Sets}^{\mathbf{F}_+}$, another choice of a "finite powerset functor" $\widetilde{K}$ is more appropriate. See [38]; also [42, p. 4].

All such functors are known to be finitary (see e.g. [38]).

Coinductive predicates in this setting can be introduced much like $\mathsf{R}\nu$ in Example 2.4 (note that $\mathbf{Sets}^{\mathbf{N}}$ is a topos), for properties like deadlock freedom. Such a language can be extended further through the modalities proposed in [34]: they correspond to constructions specific to presheaves and include the modality $\langle \overline{a}(b) \rangle$ for a binding 'input' operation. More examples will be worked out in our future paper.

Example 5.15 **(**$\mathrm{Sub}(\mathbf{Sets}^{\omega}), \mathrm{Sub}(\mathbf{Sets}^{\mathbf{I}})$**)** Consider the presheaf category $\mathbf{Sets}^{\omega}$ over the ordinal ω as a poset. The fibration $\begin{array}{c}\mathrm{Sub}(\mathbf{Sets}^{\omega})\\ \downarrow\\ \mathbf{Sets}^{\omega}\end{array}$ is finitely determined but not well-founded. It fails to satisfy Cond. iv) in Def. 3.3: let $P_n : \omega \to \mathbf{Sets}$ be the family of presheaves defined by

$$P_n(m) := \quad \big(\, 0 \text{ if } m < n; \quad 1 \text{ if } n \leq m \,\big)$$

for each $n \in \omega$. Then $P_0 > P_1 > \cdots$ is a strictly descending chain in $\mathrm{Sub}(\mathbf{y}0)$. The same counterexample works for $\mathrm{Sub}(\mathbf{Sets}^{\mathbf{I}})$.

In contrast, the subobject fibration for $\mathbf{Sets}^{\omega^{\mathrm{op}}}$ is well-founded by Lem. 5.9.

Remark 5.16 Well-foundedness fails in $\mathrm{Sub}(\mathbf{Sets}^{\omega})$, $\mathrm{Sub}(\mathbf{Sets}^{\mathbf{I}})$, and in $\mathrm{Fam}(\Omega)$ for Ω that does have a strictly descending ω^{op}-chain. This means the logics modeled by the fibrations are inherently "big." Still, extensions of our results in §3 are possible from finitary (i.e. ω-presentable) to the λ-presentable setting for bigger λ, so that they apply to the (current) nonexamples.

Acknowledgments

Thanks are due to Kazuyuki Asada, Keisuke Nakano, Keiko Nakata, Ana Sokolova, and the participants of Dagstuhl Seminar 12411 "Coalgebraic Logics" (including Samson Abramsky, Vincenzo Ciancia, Corina Cîrstea, Ernst-Erich Doberkat, Clemens Kupke, Alexander Kurz, and Yde Venema) for useful discussions. We are grateful to the anonymous referees for their careful reading and useful suggestions, too. I.H., K.C. and T.K. are supported by Grants-in-Aid for Young Scientists (A) No. 24680001 and Grants-in-Aid for Challenging Exploratory Research No. 23654033, JSPS, and by Aihara Innovative Mathematical Modeling Project, FIRST Program, JSPS/CSTP.

References

[1] S. Abramsky and V. Winschel. Coalgebraic analysis of subgame-perfect equilibria in infinite games without discounting, 2012. ArXiv:1210.4537.

[2] J. Adámek. On final coalgebras of continuous functors. *Theor. Comp. Sci.*, 294:3–29, 2003.

[3] J. Adámek, F. Bonchi, M. Hülsbusch, B. König, S. Milius and A. Silva. A coalgebraic perspective on minimization and determinization. In Birkedal [7], pp. 58–73.

[4] J. Adámek and J. Rosický. *Locally Presentable and Accessible Categories*, vol. 189 of *London Math. Soc. Lect. Note Series.* Cambridge Univ. Press, 1994.

[5] R. Atkey, N. Ghani, B. Jacobs and P. Johann. Fibrational induction meets effects. In Birkedal [7], pp. 42–57.

[6] Y. Bertot and E. Komendantskaya. Inductive and coinductive components of corecursive functions in Coq. *Elect. Notes in Theor. Comp. Sci.*, 203(5):25–47, 2008.

[7] L. Birkedal, editor. *Foundations of Software Science and Computational Structures - 15th International Conference, FOSSACS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, vol. 7213 of *Lect. Notes Comp. Sci.* Springer, 2012.

[8] F. Bonchi and D. Pous. Checking nfa equivalence with bisimulations up to congruence. In Giacobazzi and Cousot [21], pp. 457–468.

[9] J. Bradfield and C. Stirling. Modal mu-calculi. In P. Blackburn, J. van Benthem and F. Wolter, editors, *Handbook of Modal Logic*, vol. 3 of *Studies in Logic and Practical Reasoning*, chap. 12. Elsevier, 2006.

[10] C. Cîrstea. Maximal traces and path-based coalgebraic temporal logics. *Theor. Comput. Sci.*, 412(38):5025–5042, 2011.

[11] C. Cîrstea, C. Kupke and D. Pattinson. Exptime tableaux for the coalgebraic μ-calculus. In E. Grädel and R. Kahle, editors, *CSL*, vol. 5771 of *Lecture Notes in Computer Science*, pp. 179–193. Springer, 2009.

[12] C. Cîrstea, A. Kurz, D. Pattinson, L. Schröder and Y. Venema. Modal logics are coalgebraic. *Comput. J.*, 54(1):31–41, 2011.

[13] C. Cîrstea and M. Sadrzadeh. Modular games for coalgebraic fixed point logics. *Electr. Notes Theor. Comput. Sci.*, 203(5):71–92, 2008.

[14] P. Cousot and R. Cousot. Constructive versions of Tarski's fixed point theorems. *Pacific Journal of Mathematics*, 82(1):43–57, 1979.

[15] G.L. Ferrari, U. Montanari and M. Pistore. Minimizing transition systems for name passing calculi: A co-algebraic formulation. In M. Nielsen and U. Engberg, editors, *FoSSaCS*, vol. 2303 of *Lect. Notes Comp. Sci.*, pp. 129–158. Springer, 2002.

[16] G.L. Ferrari, U. Montanari and E. Tuosto. Coalgebraic minimization of hd-automata for the pi-calculus using polymorphic types. *Theor. Comp. Sci.*, 331(2–3):325–365, 2005.

[17] M. Fiore and D. Turi. Semantics of name and value passing. In *Logic in Computer Science*, pp. 93–104. IEEE, Computer Science Press, 2001.

[18] M.P. Fiore and S. Staton. Comparing operational models of name-passing process calculi. *Inf. & Comp.*, 204(4):524–560, 2006.

[19] M.P. Fiore and S. Staton. A congruence rule format for name-passing process calculi. *Inf. & Comp.*, 207(2):209–236, 2009.

[20] C. Fumex, N. Ghani and P. Johann. Indexed induction and coinduction, fibrationally. In A. Corradini, B. Klin and C. Cîrstea, editors, *CALCO*, vol. 6859 of *Lect. Notes Comp. Sci.*, pp. 176–191. Springer, 2011.

[21] R. Giacobazzi and R. Cousot, editors. *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013.* ACM, 2013.

[22] I. Hasuo. Generic forward and backward simulations II: Probabilistic simulation. In P. Gastin and F. Laroussinie, editors, *CONCUR*, vol. 6269 of *Lect. Notes Comp. Sci.*, pp. 447–461. Springer, 2010.

[23] I. Hasuo, B. Jacobs and A. Sokolova. Generic trace semantics via coinduction. *Logical Methods in Comp. Sci.*, 3(4:11), 2007.

[24] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journ. ACM*, 32(1):137–161, 1985.

[25] U. Hensel and B. Jacobs. Proof principles for datatypes with iterated recursion. In E. Moggi and G. Rosolini, editors, *Category Theory and Computer Science*, no. 1290 in Lect. Notes Comp. Sci., pp. 220–241. Springer, Berlin, 1997.

[26] C. Hermida. *Fibrations, Logical Predicates and Indeterminates.* PhD thesis, Univ. Edinburgh, 1993. Techn. rep. LFCS-93-277.

[27] C. Hermida and B. Jacobs. Structural induction and coinduction in a fibrational setting. *Inf. & Comp.*, 145:107–152, 1998.

[28] C.K. Hur, G. Neis, D. Dreyer and V. Vafeiadis. The power of parameterization in coinductive proof. In Giacobazzi and Cousot [21], pp. 193–206.

[29] B. Jacobs. *Categorical Logic and Type Theory.* North Holland, Amsterdam, 1999.

[30] B. Jacobs. Trace semantics for coalgebras. In J. Adámek and S. Milius, editors, *Coalgebraic Methods in Computer Science*, vol. 106 of *Elect. Notes in Theor. Comp. Sci.* Elsevier, Amsterdam, 2004.

[31] B. Jacobs. Predicate logic for functors and monads, March 2010. Preprint.

[32] B. Jacobs. Introduction to coalgebra. Towards mathematics of states and observations, 2012. Draft of a book (ver. 2.0), available online.

[33] P. Johnstone. *Sketches of an Elephant: A Topos Theory Compendium.* Oxford Logic Guides. Clarendon Press, 2002.

[34] B. Klin. Coalgebraic modal logic beyond **Sets**. In *MFPS XXIII*, vol. 173, pp. 177–201. Elsevier, Amsterdam, 2007.

[35] D. Kozen and N. Ruozzi. Applications of metric coinduction. *Logical Methods in Computer Science*, 5(3), 2009.

[36] C. Kupke. Terminal sequence induction via games. In P. Bosch, D. Gabelaia and J. Lang, editors, *TbiLLC*, vol. 5422 of *Lecture Notes in Computer Science*, pp. 257–271. Springer, 2007.

[37] M. Makkai and R. Paré. Accessible categories: the foundations of categorical model theory. *Contemp. Math.*, 104, 1989.

[38] M. Miculan. A categorical model of the fusion calculus. *Elect. Notes in Theor. Comp. Sci.*, 218:275–293, 2008.

[39] K. Nakata, T. Uustalu and M. Bezem. A proof pearl with the fan theorem and bar induction—walking through infinite trees with mixed induction and coinduction. In H. Yang, editor, *APLAS*, vol. 7078 of *Lecture Notes in Computer Science*, pp. 353–368. Springer, 2011.

[40] D. Pattinson. An introduction to the theory of coalgebras. Course notes for NASSLLI, 2003. Available online.

[41] J.J.M.M. Rutten. Universal coalgebra: a theory of systems. *Theor. Comp. Sci.*, 249:3–80, 2000.

[42] S. Staton. Relating coalgebraic notions of bisimulation. *Logical Methods in Comp. Sci.*, 7(1), 2011.

[43] Y. Venema. Automata and fixed point logic: A coalgebraic perspective. *Inf. Comput.*, 204(4):637–678, 2006.

[44] J. Worrell. On the final sequence of a finitary set functor. *Theor. Comp. Sci.*, 338(1–3):184–199, 2005.

A Appendix: Preliminaries

A.1 Theory of Coalgebra

Given a category $\mathbb{C}$ and an endofunctor $F : \mathbb{C} \to \mathbb{C}$, an *$F$-coalgebra* is a pair of $X \in \mathbb{C}$ and an arrow $c : X \to FX$ (we shall denote a coalgebra simply by $X \xrightarrow{c} FX$). The notion has turned out to be a useful categorical abstraction of state-based dynamic systems. In an F-coalgebra $X \xrightarrow{c} FX$, the *carrier object* $X \in \mathbb{C}$ is understood as a state space; the functor F specifies the *behavior type*; and the arrow c represents actual dynamics. In the most common setting of $\mathbb{C} = \mathbf{Sets}$, examples of functors F (and the corresponding behavior types) are:

- $A \times (\_)$ for A-stream automata;
- $\mathcal{P}(\mathsf{AP}) \times \mathcal{P}(\_)$ for Kripke models;
- $\mathcal{P}(\mathsf{AP}) \times \mathcal{P}_\omega(\_)$ for finitely branching Kripke models, with where $\mathcal{P}_\omega$ is the finite powerset functor;
- $\mathcal{P}(A \times \_)$ for labeled transition systems;
- $\mathcal{D}(A \times \_)$ for generative probabilistic systems;

and so on. See [32, 41] for detailed introduction.

In the theory of coalgebra as a categorical theory of (state-based dynamical) systems, the notion of *final coalgebra* plays a prominent role. A final F-coalgebra $Z \xrightarrow{\zeta} FZ$ is one such that, for any F-coalgebra $X \xrightarrow{c} FX$, there is a unique morphism of coalgebras from c to ζ.

$$\begin{tikzcd} FX \arrow[r, dashed, "F\overline{c}"] & FZ \\ X \arrow[u, "c"] \arrow[r, dashed, "\overline{c}"] & Z \arrow[u, "\text{final}"', "\zeta"] \end{tikzcd} \tag{A.1}$$

Its system-theoretic significance is that: 1 Z is often the collection of "all possible F-behaviors"; and 2 the induced arrow $\overline{c}$ assigns, to each state in X, its behavior. The "behaviors" here follow a black-box view on systems (it ignores internal states) and often captures the natural notion of "F-bisimilarity."

Therefore a question arises if a final F-coalgebra exists. The well-known Lambek lemma (that ζ is necessarily an iso) prohibits e.g. a final $\mathcal{P}$-coalgebra. What matters here is the *size* of F: when it is suitably bounded, a concrete construction of a final coalgebra is known. It obtains a final coalgebra via a *final F-sequence* (Here 1 is a final object in $\mathbb{C}$).

$$\begin{tikzcd} 1 & F1 \arrow[l, "!"'] & \cdots \arrow[l] & F^i 1 \arrow[l, "F^{i-1}\,!"'] & \cdots \arrow[l, "F^i\,!"'] \end{tikzcd} \tag{A.2}$$

In particular, if F is *finitary* (a size restriction described later), a final coalgebra arises as a suitable quotient of the limit of the final sequence (3). This construction in **Sets** is worked out in [44]; it is further extended to locally presentable categories (those are categories suited for speaking of "size") with additional assumptions in [2]. The current paper's goal is to apply this construction also to coinductive predicates.

A.2 Locally Finitely Presentable Categories

The theory of coalgebra has been mainly developed in the base category $\mathbb{C} = \mathbf{Sets}$. Exceptions include the category of nominal sets or (pre)sheaf categories (e.g. [18,19]) for name-passing calculi, and Kleisli categories (e.g. [22,23]) for trace semantics and simulation. The current paper follows [2,34] and finds locally finitely presentable categories a convenient abstract setting. Here we follow [4] and list a minimal set of definitions and results on locally finitely presentable categories.

The following is a categorical formalization of "finiteness" of objects. Examples are finite sets (in **Sets**), and algebras presented by finitely many generators and finitely many equations (in suitable categories of algebras).

Definition A.1 (Finitely presentable object) An object $X \in \mathbb{C}$ is *finitely presentable (FP)* if the functor $\mathbb{C}(X, \_\,) : \mathbb{C} \to \mathbf{Sets}$ preserves filtered colimits.

Definition A.2 (Locally finitely presentable category) A category $\mathbb{C}$ is *locally finitely presentable (LFP)* if it is cocomplete and it has a (small) set $\mathbb{F}$ of FP objects such that every object is a directed colimit of objects in $\mathbb{F}$.

Lemma A.3 *Let $\mathbb{C}$ be LFP, with a set $\mathbb{F}$ of FP objects as in Def. 3.1; and $X \in \mathbb{C}$. The* canonical diagram *for X with respect to $\mathbb{F}$*

$$(\mathbb{F}{\downarrow}X) \xrightarrow{\pi} \mathbb{F} \hookrightarrow \mathbb{C} \tag{A.3}$$

has X as its colimit. Here π is the projection.

Proof The proof of [4, Prop. 1.22] yields the claim. □

Lemma A.4 [4, Cor. 1.28 & Prop. 1.61] *Let $\mathbb{C}$ be LFP.*

(i) *$\mathbb{C}$ is complete.*

(ii) *$\mathbb{C}$ has* (StrongEpi, Mono)- *and* (Epi, StrongMono)*-factorization structures.* □

The following notion (which is already in Def. A.1) is about the "size" of functors. An intuition (when $\mathbb{C} = \mathbf{Sets}$) is: a functor F is finitary if F's action FX on an arbitrary set X is determined by its action FX' on all the finite subsets $X' \subseteq X$.

Definition A.5 (Finitary functor) An endofunctor $F : \mathbb{C} \to \mathbb{C}$ is *finitary* if it preserves filtered colimits.

This notion is commonly used to bound the "branching degree" of systems as F-coalgebras. For example, the finite powerset functor $\mathcal{P}_\omega$ is finitary; the (full) powerset functor $\mathcal{P}$ is not.

There are many LFP categories, among which are **Sets**, the category **Posets** of posets and monotone maps, and categories of algebras with finitary operations. See [4] for more examples.

Example A.6 (Presheaf categories) Let $\mathbb{A}$ be a small category. The presheaf category $\mathbf{Sets}^{\mathbb{A}}$ is LFP: the set

$$\mathbb{F} := \{\text{finite colimits of representable presheaves } \mathbf{y}A\} \ ,$$

where $\mathbf{y}A = \mathbb{A}(A, \_)$, satisfies the conditions of Def. A.1.

Definition A.7 (Finitely generated object) An object $X \in \mathbb{C}$ is *finitely generated (FG)* if the functor $\mathbb{C}(X, \_) : \mathbb{C} \to \mathbf{Sets}$ preserves directed colimits of monos—that is, directed colimits of diagrams in which every (connecting) arrow is a mono.

It is clear that FP implies FG. In algebraic terms, FP objects are algebras presented by finitely many generators and finitely many equations; while for FG objects only a set of generators is required to be finite. The two notions coincide in "non-algebraic" examples such as **Sets**. See [4, §1.E].

A.3 Fibrations

We follow [29], although we focus on the simpler notion of poset fibration.

Introduction (via Indexed Posets)

This paper's interest is in coinductive predicates, hence in predicate logic. The most straightforward formalization of predicate is as a subset $P \subseteq X$ of a set (a 'universe') X: an element $x \in X$ *satisfies* P if $x \in P$. Accompanying is the natural notion of entailment: P *entails* Q if $P \subseteq Q$. This way we obtain the poset $(2^X, \subseteq)$ of predicates over X.

However it is not on a single universe X that we consider predicates. For example, in a situation where there are two Kripke models $c = (X, \to, V_X)$, $d = (Y, \to, V_Y)$ and a "homomorphism" $f : X \to Y$, a natural question is if the interpretation of a formula $\nu u.\alpha$ is preserved by f. (It is; see Prop. 3.11). Here we are comparing the predicate $[\![\nu u.\alpha]\!]_c \subseteq X$ with the predicate $[\![\nu u.\alpha]\!]_d \subseteq Y$ *reindexed* via $f : X \to Y$. The latter is concretely described as the *inverse image*

$$f^{-1}\big([\![\nu u.\alpha]\!]_d\big) \;=\; \big\{\, x \in X \,\big|\, f(x) \in [\![\nu u.\alpha]\!]_d \,\big\} \ .$$

Therefore a reindexing structure is also relevant to predicate logic: a function $f : X \to Y$ induces reindexing $f^{-1} : 2^Y \to 2^X$. Additionally, the map f^{-1} is monotone.

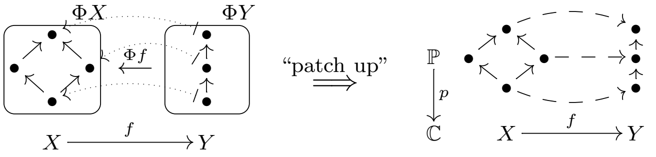
To summarize: 1) predicates on a universe X form a poset; 2) a function $f : X \to Y$ between universes induces a monotone reindexing function from the collection of predicates over X to that over Y. Such a situation is nicely described as a (contravariant) functor

$$\Phi \;:\; \mathbb{C}^{\mathrm{op}} \longrightarrow \mathbf{Posets} \ , \tag{A.4}$$

where **Posets** is the category of posets and monotone functions. The functor Φ assigns, to each 'universe' $X \in \mathbb{C}$, the poset ΦX of predicates over X. Moreover,

$f : X \to Y$ in $\mathbb{C}$ induces a reindexing map $\Phi f : \Phi Y \to \Phi X$. This functor Φ is a special case of an *indexed category* [29, §1.10].

In the current paper, however, we favor an equivalent presentation of such a structure by a *fibration*, since we find the latter to be more amenable to generalization of structures in ordinary category theory (such as limits). The equivalence between index categories and fibrations are well-known; here we sketch the *Grothendieck construction* from the former to the latter. Its idea is to "patch up" the posets $(\Phi X)_{X \in \mathbb{C}}$ and form a big category $\mathbb{P}$, as in the following figure.



On the right we add some arrows (denoted by $\dashrightarrow$) so that we have an arrow $(\Phi f)(Q) \to Q$ in $\mathbb{P}$ for each $Q \in \Phi Y$. (On the left the arrows $\mapsto$ depicts the action of the map Φf.) The above diagram in $\mathbb{P}$ should be understood as a Hasse diagram: those arrows which arise from composition are not depicted.

Formally:

Definition A.8 (The Grothendieck construction) Given $\Phi : \mathbb{C}^{\mathrm{op}} \to \mathbf{Posets}$, we define the category $\mathbb{P}_\Phi$ by

- its object is a pair (X, P) of an object $X \in \mathbb{C}$ and an element P of the poset ΦX; and
- its arrow $f : (X, P) \to (Y, Q)$ is an arrow $f : X \to Y$ in $\mathbb{C}$ such that
$$P \leq (\Phi f)(Q) \enspace .$$
Here $\leq$ refers to the order of ΦX.

Thus arises a category $\mathbb{P}_\Phi$ that incorporates: the order structure of each of the posets $(\Phi X)_{X \in \mathbb{C}}$; and the reindexing structure by $(\Phi f)_{f\colon \mathbb{C}\text{-arrow}}$. For fixed $X \in \mathbb{C}$, the objects of the form (X, P) and the arrows id_X between them form a subcategory of $\mathbb{P}$. This is denoted by $\mathbb{P}_X$ and called the *fiber* over X. It is obvious that $\mathbb{P}_X$ is a poset that is isomorphic to ΦX.

Moreover, there is a canonical projection functor $p : \mathbb{P} \to \mathbb{C}$ that carries (X, P) to X.

Formal Definition of (Poset) Fibration

We axiomatize those structures which arise in the way described above.

Definition A.9 ((Poset) fibration) A *(poset) fibration* $\begin{array}{c}\mathbb{P}\\ \downarrow p\\ \mathbb{C}\end{array}$ consists of two categories $\mathbb{P}, \mathbb{C}$ and a functor $p : \mathbb{P} \to \mathbb{C}$, that satisfy the following properties.

- Each fiber $\mathbb{P}_X$ is a poset. Here the *fiber* $\mathbb{P}_X$ for $X \in \mathbb{C}$ is the subcategory of $\mathbb{P}$ consisting of objects $P \in \mathbb{P}$ such that $pP = X$ and arrows $f : P \to Q$ such that $pf = \mathrm{id}_X$ (such arrows are said to be *vertical*).

- Given $f : X \to Y$ in $\mathbb{C}$ and $Q \in \mathbb{P}_Y$, there is an object $f^*Q \in \mathbb{P}_X$ and a $\mathbb{P}$-arrow $\overline{f}Q : f^*Q \to Q$ with the following universal property. For any $P \in \mathbb{P}_X$ and $g : P \to Q$ in $\mathbb{P}$, if $pg = f$ then g factors through $\overline{f}(Q)$ uniquely via a vertical arrow. That is, there exists a unique g' such that $g = \overline{f}(Q) \circ g'$ and $pg' = \mathrm{id}_X$.

$$\begin{tikzcd}
\mathbb{P} \arrow[dd, "p"] & & Q & & f^*Q \arrow[r, "\overline{f}(Q)"] & Q \\
& & & \Longrightarrow & P \arrow[u, dotted, "g'"] \arrow[ur, "g"'] & \\
\mathbb{C} & X \arrow[r, "f"] & Y & & X \arrow[r, "f"] & Y
\end{tikzcd}$$

- The correspondences $(\_)^*$ and $\overline{(\_)}$ are functorial:

$$\begin{aligned}
\mathrm{id}_Y^* Q &= Q \ , & (g \circ f)^*(Q) &= f^*(g^*Q) \ , \\
\overline{\mathrm{id}_Y}(Q) &= \mathrm{id}_Q \ , & \overline{g \circ f}(Q) &= \overline{g}Q \circ \overline{f}(g^*Q) \ .
\end{aligned}$$

The last equality can be depicted as follows.

$$\begin{tikzcd}
\mathbb{P} \arrow[dd, "p"] & f^*(g^*Q) \arrow[r, "\overline{f}(g^*Q)"] \arrow[d, phantom, "\shortparallel"] & g^*Q \arrow[r, "\overline{g}Q"] & Q \\
& (g \circ f)^*Q \arrow[urr, bend right=10, "\overline{g \circ f}(Q)"'] & & \\
\mathbb{C} & X \arrow[r, "f"] & Y \arrow[r, "g"] & Z
\end{tikzcd}$$

The category $\mathbb{P}$ is called the *total category* of the fibration; $\mathbb{C}$ is the *base category*. The arrow $\overline{f}Q : f^*Q \to Q$ is called the *Cartesian lifting* of f and Q. An arrow in $\mathbb{P}$ is *Cartesian* (or *reindexing*) if it coincides with $\overline{f}Q$ for some f and Q.

In the case where $\begin{array}{c}\mathbb{P}\\ \downarrow p\\ \mathbb{C}\end{array}$ is induced by an indexed category $\Phi : \mathbb{C}^{\mathrm{op}} \to \mathbf{Posets}$ via Def. A.8, a Cartesian lifting is obviously given by $f^*(Q) = (\Phi f)(Q)$.

In the current paper we focus on poset fibrations (which we shall simply call *fibrations*). In a (general) fibration a fiber $\mathbb{P}_X$ is not just a poset but a category, and this elicits a lot of technical subtleties. Nevertheless, it should not be hard to generalize the current paper's results to general, not necessarily poset, fibrations (especially to the split ones).

We shall often denote a vertical arrow in $\mathbb{P}$ (i.e. an arrow inside a fiber) by $\leq$.

Examples

Example A.10 (Subobject fibration) Let $\mathbb{C}$ be a (well-powered) category with finite limits. The category $\mathrm{Sub}(\mathbb{C})$ is defined by: its object is a pair (P, X) of $X \in \mathbb{C}$ and its subobject $P \rightarrowtail X$ (we write $(P \rightarrowtail X) \in \mathrm{Sub}(\mathbb{C})$); and its arrow $(P \rightarrowtail X) \xrightarrow{f} (V \rightarrowtail Y)$ is a $\mathbb{C}$-arrow $f : X \to Y$ that restricts to $P \to Q$. That is, given an arrow $f : X \to Y$ in $\mathbb{C}$,

$$\begin{array}{c} f \text{ is an arrow in } \mathrm{Sub}(\mathbb{C}) \\ (P \overset{m}{\rightarrowtail} X) \xrightarrow{f} (Q \overset{n}{\rightarrowtail} Y) \end{array} \iff \exists f' \text{ s.t. } \begin{tikzcd} P \arrow[r, dashed, "f'"] \arrow[d, tail, "m"'] & Q \arrow[d, tail, "n"] \\ X \arrow[r, "f"'] & Y \end{tikzcd} . \tag{A.5}$$

The projection $(P \rightarrowtail X) \mapsto X$ defines a functor; thus arises the *subobject fibration* $\begin{array}{c}\mathrm{Sub}(\mathbb{C})\\ \downarrow\\ \mathbb{C}\end{array}$ of $\mathbb{C}$. In particular, given $X \xrightarrow{f} Y$ in $\mathbb{C}$ and $(Q \rightarrowtail Y) \in \mathrm{Sub}(Y)$, the Cartesian lifting f^*Q is defined by a pullback.

$$\begin{tikzcd}
f^*Q \arrow[r, dashed, "\overline{f}Q"] \arrow[d, tail, "m"'] \arrow[dr, phantom, "\lrcorner", very near start] & Q \arrow[d, tail, "n"] \\
X \arrow[r, "f"'] & Y
\end{tikzcd}$$

A special case is the following most straightforward modeling of predicate logic. It arises from the contravariant powerset functor $2^{(\_)} : \mathbf{Sets}^{\mathrm{op}} \to \mathbf{Posets}$ via Def. A.8.

Example A.11 ($\begin{array}{c}\mathbf{Pred}\\ \downarrow\\ \mathbf{Sets}\end{array}$) The subobject fibration $\begin{array}{c}\mathrm{Sub}(\mathbf{Sets})\\ \downarrow\\ \mathbf{Sets}\end{array}$ of **Sets** is denoted by $\begin{array}{c}\mathbf{Pred}\\ \downarrow\\ \mathbf{Sets}\end{array}$. An object of its total category is often denoted by $(U \subseteq X)$. Reindexing is given by inverse images.

More concretely, in the category **Pred**, an object is a pair (P, X) of a set X and its subset $P \subseteq X$; an arrow $(P \subseteq X) \xrightarrow{f} (Q \subseteq Y)$ is a function $X \xrightarrow{f} Y$ that restricts to $P \to Q$ (i.e. $P \subseteq f^{-1}Q$).

Example A.12 (Rel) The fibration $\begin{array}{c}\mathbf{Rel}\\ \downarrow\\ \mathbf{Sets}\end{array}$ can be introduced from $\begin{array}{c}\mathbf{Pred}\\ \downarrow\\ \mathbf{Sets}\end{array}$ via the following change-of-base.

$$\begin{tikzcd}
\mathbf{Rel} \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & \mathbf{Pred} \arrow[d] \\
\mathbf{Sets} \arrow[r, "X \mapsto X \times X"'] & \mathbf{Sets}
\end{tikzcd}$$

Concretely, an object of **Rel** is a pair (X, R) of a set X and a relation $R \subseteq X \times X$; an arrow $f : (X, R) \to (Y, S)$ is a function $f : X \to Y$ such that xRx' implies $f(x)Sf(x')$. See [29, p. 14].

Example A.13 (Family fibration) The *family fibration* $\begin{array}{c}\mathrm{Fam}(\Omega)\\ \downarrow\\ \mathbf{Sets}\end{array}$ over a poset Ω is introduced as follows. An object in the fiber $\mathrm{Fam}(\Omega)_X$ is a function $f : X \to \Omega$; and an arrow $(X \xrightarrow{f} \Omega) \xrightarrow{k} (Y \xrightarrow{g} \Omega)$ in the total category $\mathrm{Fam}(\Omega)$ is a function $k : X \to Y$ such that $f(x) \leq g(k(x))$ for each $x \in X$. See e.g. [29, Def. 1.2.1] for more details.

Structures in a Fibration

In a fibration $\begin{array}{c}\mathbb{P}\\ \downarrow p\\ \mathbb{C}\end{array}$, a $\mathbb{C}$-arrow $X \xrightarrow{f} Y$ induces a correspondence $\mathbb{P}_Y \xrightarrow{f^*} \mathbb{P}_X$ via reindexing. This is easily seen to be a monotone map (i.e. a functor between posets as categories).

Definition A.14 (Fiberwise (co)limits) A fibration $\begin{array}{c}\mathbb{P}\\ \downarrow p\\ \mathbb{C}\end{array}$ is said to have *fiberwise limits* if:

- each fiber $\mathbb{P}_X$ has, as a category, all limits (meaning it has arbitrary inf's $\bigwedge$); and
- for each $\mathbb{C}$-arrow $X \xrightarrow{f} Y$, the reindexing functor $\mathbb{P}_Y \xrightarrow{f^*} \mathbb{P}_X$ preserves these limits.

In this case each fiber $\mathbb{P}_X$ has a final object (denoted by $\top_X$).

Similarly, a fibration has *fiberwise colimits* if each fiber has them and they are preserved by reindexing.

The following notions must be distinguished from "fiberwise (co)products."

Definition A.15 ((Co)products between fibers) A fibration $\begin{array}{c}\mathbb{P}\\ \downarrow p\\ \mathbb{C}\end{array}$ is said to have *products (between fibers)* if

- each reindexing functor $f^* : \mathbb{P}_Y \to \mathbb{P}_X$ has a right adjoint $f^* \dashv \prod_f$; and
- the functors $(\prod_f)_f$ satisfy the so-called *Beck-Chevalley condition*. See [29, §1.9].

Similarly, a fibration has *coproducts (between fibers)* if each reindexing has a left adjoint $\coprod_f$ and they satisfy the Beck-Chevalley condition.

The prototype example $\begin{array}{c}\mathbf{Pred}\\ \downarrow\\ \mathbf{Sets}\end{array}$ has fiberwise (co)limits: each fiber is a complete lattice; and $\bigwedge$ and $\bigvee$ are preserved by inverse images. It has (co)products $\coprod$ between fibers, too: specifically $\coprod_f$ is given by the *direct image* of the function f. See [29, §1.9].

Throughout the paper we rely on the following result. It follows from [29, Lem. 9.1.2 & Prop. 9.2.1], and extends Lem. 3.5.

Lemma A.16 *Let* $\begin{array}{c}\mathbb{P}\\ \downarrow p\\ \mathbb{C}\end{array}$ *be a fibration. Assume that* $\mathbb{C}$ *is complete; then the following are equivalent.*

(i) *The fibration* p *has fiberwise limits.*

(ii) *The total category* $\mathbb{P}$ *is complete and* $p : \mathbb{P} \to \mathbb{C}$ *preserves limits.*

If this is the case, a limit of a small diagram $(P_I)_{I\in\mathbb{I}}$ *in* $\mathbb{P}$ *can be given by*

$$\textstyle\bigwedge_{I\in\mathbb{I}}(\pi_I^* P_I) \qquad \textit{over } \mathrm{Lim}_{I\in\mathbb{I}}\, X_I\,.$$

Here $X_I := pP_I$*;* $(\mathrm{Lim}_{I\in\mathbb{I}}\, X_I \xrightarrow{\pi_I} X_I)_{I\in\mathbb{I}}$ *is a limiting cone in* $\mathbb{C}$*; and* $\bigwedge_{I\in\mathbb{I}}$ *denotes the limit computed in the fiber* $\mathbb{P}_{\mathrm{Lim}_I X_I}$*.*

(Sort of) dually, let $\begin{array}{c}\mathbb{P}\\ \downarrow p\\ \mathbb{C}\end{array}$ *be a fibration with coproducts* $\coprod$ *between fibers, and assume that* $\mathbb{C}$ *is cocomplete. Then* p *has fiberwise colimits if and only if* $\mathbb{P}$ *is cocomplete and* $p : \mathbb{P} \to \mathbb{C}$ *preserves colimits. In this case a colimit of a small diagram* $(P_I)_{I\in\mathbb{I}}$ *in* $\mathbb{P}$ *can be given by*

$$\textstyle\bigvee_{I\in\mathbb{I}}(\coprod_{\kappa_I} P_I) \qquad \textit{over } \mathrm{Colim}_I\, X_I\,,$$

where $X_I := pP_I$ *and* $(X_I \xrightarrow{\kappa_I} \mathrm{Colim}_I\, X_I)_{I\in\mathbb{I}}$ *is a colimiting cocone in* $\mathbb{C}$*.* □

B Appendix: Omitted Proofs

B.1 Proof of Lem. 3.6

Proof We proceed by steps.

a) We observe that, in Fig. 1, the top diagram is carried to the one below by the functor $p : \mathbb{P} \to \mathbb{C}$. This is straightforward: the arrow $\varphi\top_1 \to \top_1$ must be carried to the unique arrow $! : F1 \dashrightarrow 1$; on the mediating arrow b' in $\mathbb{P}$, since pb' is again a mediating arrow in $\mathbb{C}$, it must coincide with b.

b) Before moving on, we observe that Cond. iii) in Def. 3.2 yields a seemingly stronger statement (Cond. iii') below).

Sublemma B.1 *For a finitely determined fibration* $\begin{array}{c}\mathbb{P}\\ \downarrow p\\ \mathbb{C}\end{array}$ *the following holds.*

iii') Let $X \in \mathbb{C}$; $P, Q \in \mathbb{P}_X$; and $(Y_J)_{J\in\mathbb{J}}$ be an arbitrary filtered diagram in $\mathbb{C}$ such that $\mathrm{Colim}_J\, Y_J = X$, with a colimiting cocone $(Y_J \overset{\gamma_J}{\to} X)_{J\in\mathbb{J}}$. Then $P \le Q$ if and only if for each $J \in \mathbb{J}$, $\gamma_J^* P \le \gamma_J^* Q$ in $\mathbb{P}_{Y_J}$.

Proof (Of Sublem. B.1) The only nontrivial statement is the 'if' part of the direction iii) $\Rightarrow$ iii'). It suffices to show that $\gamma_J^* P \le \gamma_J^* Q$ (for each $J \in \mathbb{J}$) implies $\kappa_I^* P \le \kappa_I^* Q$ (for each $I \in \mathbb{I}$), where κ_I and $\mathbb{I}$ are as in Cond. iii).

Let $I \in \mathbb{I}$. Since X_I is FP, an arrow $\kappa_I : X_I \to X$ to a filtered colimit $X = \mathrm{Colim}_J\, Y_J$ factors through some $Y_{J_I} \overset{\gamma_{J_I}}{\to} X$, as in the diagram below.

$$\begin{tikzcd}
X_I \arrow[rr, "\kappa_I"] \arrow[r, "h_I"'] & Y_{J_I} \arrow[r, "\gamma_{J_I}"'] & X = \mathrm{Colim}_J\, Y_J
\end{tikzcd}$$

Now we have $\kappa_I^* P = h_I^* \gamma_{J_I}^* P \le h_I^* \gamma_{J_I}^* Q = \kappa_I^* Q$, where the inequality is by the assumption that $\gamma_J^* P \le \gamma_J^* Q$ for each $J \in \mathbb{J}$. This proves Sublem. B.1. □

c) By Step a) we see that $\varphi^{\omega+1}\top_1 \le b^*(\varphi^\omega \top_1)$ by the universality of a Cartesian arrow. In what follows we shall prove its converse:

$$b^*(\varphi^\omega\top_1) \le \varphi^{\omega+1}\top_1 \qquad \text{in } \mathbb{P}_{F^{\omega+1}1}. \tag{B.1}$$

Let us take a directed diagram $(X_I)_{I\in\mathbb{I}}$ in $\mathbb{C}$ such that $X_I \in \mathbb{F}$ (for each $I \in \mathbb{I}$) and $F^\omega 1 = \mathrm{Colim}_{I\in\mathbb{I}}\, X_I$, with $(X_I \overset{\kappa_I}{\to} F^\omega 1)_{I\in\mathbb{I}}$ being the colimiting cocone. Then we have

$$F^{\omega+1}1 = F(\operatorname*{Colim}_{I\in\mathbb{I}} X_I) = \operatorname*{Colim}_{I\in\mathbb{I}} FX_I \enspace ,$$

by the assumption that F is finitary; moreover $(FX_I \overset{F\kappa_I}{\to} F^{\omega+1}1)_{I\in\mathbb{I}}$ is a colimiting cocone. The diagram $(X_I)_{I\in\mathbb{I}}$ is directed, and so is the latter diagram $(FX_I)_{I\in\mathbb{I}}$. Thus by Cond. iii') in Sublem. B.1, showing the following proves (B.1).

$$(F\kappa_I)^*\Bigl(b^*(\varphi^\omega\top_1)\Bigr) \le (F\kappa_I)^*(\varphi^{\omega+1}\top_1) \quad \text{for each } I \in \mathbb{I}. \tag{B.2}$$

d) To prove (B.2) we first prove the following fact: for each $I \in \mathbb{I}$ there exists $i_I \in \omega$ such that

$$\kappa_I^*(\varphi^\omega\top_1) = \kappa_I^*\bigl(\,\pi_{i_I}^*(\varphi^{i_I}\top_1)\bigr) \quad \text{in } \mathbb{P}_{X_I}. \tag{B.3}$$

That is: the final sequence in $\mathbb{P}$ (Fig. 1), when restricted to X_I (that is FP), stabilizes within finitely many steps. Indeed, by Lem. A.16 the limit $\varphi^\omega\top_1$ is described as an inf in $\mathbb{P}_{F^\omega 1}$:

$$\varphi^\omega\top_1 \;=\; \textstyle\bigwedge_{i\in\omega} \pi_i^*(\varphi^i\top_1) \enspace . \tag{B.4}$$

Therefore we have $\kappa_I^*(\varphi^\omega\top_1) = \bigwedge_{i\in\omega} \kappa_I^*\pi_i^*(\varphi^i\top_1)$ since reindexing κ_I^* preserves fiberwise limits $\bigwedge$. Now we claim that the sequence $\bigl(\kappa_I^*\pi_i^*(\varphi^i\top_1)\bigr)_{i\in\omega}$ in $\mathbb{P}_{X_I}$ is descending: it follows from the fact that $\bigl(\pi_i^*(\varphi^i\top_1)\bigr)_{i\in\omega}$ in $\mathbb{P}_{F^\omega 1}$ is descending, which in turn is shown from the universality of the Cartesian arrow $\overline{\pi_i}(\varphi^i\top_1)$. See below.

$$\begin{tikzcd}
 & & & \pi_i^*(\varphi^i\top_1) \arrow[ddll, bend right, "\overline{\pi_i}(\varphi^i\top_1)"'] \\
\mathbb{P} \arrow[ddd] & & & \pi_{i+1}^*(\varphi^{i+1}\top_1) \arrow[u, "!"] \arrow[dl] \\
 & \varphi^i\top_1 & \varphi^{i+1}\top_1 \arrow[l] & \\
 & & & F^\omega 1 \arrow[dll, bend right, "\pi_i"'] \arrow[dl, "\pi_{i+1}"] \\
\mathbb{C} & F^i 1 & F^{i+1} 1 \arrow[l, "F^i\,!"'] &
\end{tikzcd}$$

Therefore, by p being a well-founded fibration (Def. 3.3), there exists $i_I \in \omega$ at which the descending sequence stabilizes, that is,

$$\textstyle\bigwedge_{i\in\omega} \pi_i^*(\varphi^i\top_1) = \pi_{i_I}^*(\varphi^{i_I}\top_1) \quad \text{in } \mathbb{P}_{F^\omega 1}.$$

Combined with (B.4), this proves (B.3).

e) Finally let us prove (B.2). For each $I \in \mathbb{I}$,

$$\begin{aligned}
&(F\kappa_I)^*\bigl(b^*(\varphi^\omega\top_1)\bigr)\\
&= (F\kappa_I)^*\bigl(b^*\bigl(\textstyle\bigwedge_{i\in\omega}\pi_i^*(\varphi^i\top_1)\bigr)\bigr) \quad \text{by (B.4)}\\
&= \textstyle\bigwedge_{i\in\omega}(F\kappa_I)^*\bigl(b^*\bigl(\pi_i^*(\varphi^i\top_1)\bigr)\bigr) \quad \text{reindexing preserves } \bigwedge\\
&= \textstyle\bigwedge_{i\in\omega}(F\kappa_I)^*\bigl((F\pi_{i-1})^*(\varphi^i\top_1)\bigr)\\
&\qquad\qquad\qquad\qquad \text{by } \pi_i \circ b = F\pi_{i-1} \text{ (see Fig. 1)}\\
&= \textstyle\bigwedge_{i\in\omega}\varphi\bigl((\pi_{i-1}\circ\kappa_I)^*(\varphi^{i-1}\top_1)\bigr) \quad \text{by Def. 2.2}\\
&\le \varphi\bigl((\pi_{i_I}\circ\kappa_I)^*(\varphi^{i_I}\top_1)\bigr) \quad \text{letting } i = i_I + 1 \text{ on the LHS}\\
&= \varphi\bigl(\kappa_I^*\pi_{i_I}^*(\varphi^{i_I}\top_1)\bigr) = \varphi\bigl(\kappa_I^*(\varphi^\omega\top_1)\bigr) \quad \text{by (B.3)}\\
&= (F\kappa_I)^*(\varphi^{\omega+1}\top_1)\\
&\qquad\qquad \text{by Def. 2.2 and } \varphi^{\omega+1}\top_1 = \varphi(\varphi^\omega\top_1).
\end{aligned}$$

This proves (B.2) and concludes the proof of Lem. 3.6. □

B.2 Proof of Thm. 3.7

Proof We proceed by steps.

ia) We first show that $c_\omega^*(\varphi^\omega\top_1)$ indeed carries a $(c^*\circ\varphi)$-coalgebra.

$$
\begin{aligned}
&c^*\bigl(\varphi(c_\omega^*(\varphi^\omega\top_1))\bigr)\\
&= c^*\bigl((Fc_\omega)^*(\varphi(\varphi^\omega\top_1))\bigr) \quad \text{by Def. 2.2}\\
&= c^*\bigl((Fc_\omega)^*(b^*(\varphi^\omega\top_1))\bigr) \quad \text{by Lem. 3.6}\\
&= (b\circ Fc_\omega\circ c)^*(\varphi^\omega\top_1)\\
&= c_\omega^*(\varphi^\omega\top_1) \enspace .
\end{aligned}
$$

For the last equality we used $b\circ Fc_\omega\circ c = c_\omega$, which is proved by showing that $b\circ Fc_\omega\circ c$ is also a mediating map in (7). Indeed, for each $i\in\omega$,

$$
\begin{aligned}
&\pi_i\circ b\circ Fc_\omega\circ c\\
&= F\pi_{i-1}\circ Fc_\omega\circ c \quad \text{see Fig. 1}\\
&= Fc_{i-1}\circ c \quad \text{by (7)}\\
&= c_i \quad \text{by def. of } c_i.
\end{aligned}
$$

ib) We show that the coalgebra obtained in Step a) is final. Let $U\le c^*(\varphi U)$ be an arbitrary $(c^*\circ\varphi)$-coalgebra (i.e. a φ-invariant in c), where $U\in\mathbb{P}_X$. We aim to establish the following diagram in $\mathbb{P}$ and see that it is above the one in (7).

$$
\begin{tikzcd}
 & & & & & \varphi^\omega\top_1 \arrow[dlllll, bend right] \arrow[dllll, bend right] \arrow[dll, bend right] \\
\top_1 & \varphi\top_1 \arrow[l] & \cdots \arrow[l] & \varphi^i\top_1 \arrow[l] & \cdots \arrow[l] & \\
 & & & & & U \arrow[ulllll, bend left] \arrow[ullll, bend left] \arrow[ull, bend left] \arrow[uu, dashed, bend right, "c'_\omega"']
\end{tikzcd}
\tag{B.5}
$$

We first show that

$$
U\le c_i^*(\varphi^i\top_1) \qquad \text{for each } i\in\omega. \tag{B.6}
$$

The proof is by induction. The base case $i=0$ is obvious since reindexing c_i^* preserves $\top$. For the step case:

$$
\begin{aligned}
&U\le c^*(\varphi U) \quad U \text{ carries a } (c^*\circ\varphi)\text{-coalgebra}\\
&\le c^*\bigl(\varphi\bigl(c_i^*(\varphi^i\top_1)\bigr)\bigr) \quad \text{by induction hypothesis}\\
&= c^*\bigl((Fc_i)^*(\varphi^{i+1}\top_1)\bigr) \quad \text{by Def. 2.2}\\
&= (c_{i+1})^*(\varphi^{i+1}\top_1) \quad \text{by def. of } c_{i+1}.
\end{aligned}
$$

This proves (B.6) and establishes the arrows $U\to\varphi^i\top_1$ in (B.5), for each i. Therefore we obtain a mediating map $c'_\omega : U \dashrightarrow \varphi^\omega\top_1$ to the limit $\varphi^\omega\top_1$, too. The arrow c'_ω is easily shown to be above c_ω (much like b' in Fig. 1 is shown to be above b); this means $U\le c_\omega^*(\varphi^\omega\top_1)$. Since $\mathbb{P}_X$ is a poset, this arrow $\le$ is necessarily a coalgebra morphism from U to $c_\omega^*(\varphi^\omega\top_1)$; moreover it is a unique such. This proves i).

ii) We have

$$\begin{aligned}
&[\![\nu\varphi]\!]_c \\
&= c_\omega^*(\varphi^\omega \top_1) \quad \text{by i)} \\
&= c_\omega^*\bigl(\textstyle\bigwedge_{i\in\omega} \pi_i^*(\varphi^i \top_1)\bigr) \quad \text{by Lem. A.16} \\
&= \textstyle\bigwedge_{i\in\omega} c_\omega^*\bigl(\pi_i^*(\varphi^i \top_1)\bigr) \quad \text{since reindexing preserves } \bigwedge \\
&= \textstyle\bigwedge_{i\in\omega} c_i^*(\varphi^i \top_1) \quad \text{by def. of } c_\omega.
\end{aligned} \tag{B.7}$$

Furthermore, $c_i^*(\varphi^i \top_1)$ in the above is seen to be equal to $(c^* \circ \varphi)^i(\top_X)$. This is shown by induction on $i \in \omega$. For $i = 0$ the claim amounts to $!^*(\top_1) = \top_X$, which holds since reindexing preserves $\top$. For the step case,

$$\begin{aligned}
&c_{i+1}^*(\varphi^{i+1} \top_1) \\
&= c^*(Fc_i)^*(\varphi^{i+1} \top_1) \quad \text{by } c_{i+1} = Fc_i \circ c \\
&= c^*\varphi\bigl(c_i^*(\varphi^i \top_1)\bigr) \quad \text{by Def. 2.2} \\
&= (c^* \circ \varphi)\bigl(\,(c^* \circ \varphi)^i(\top_X)\,\bigr) \quad \text{by induction hypothesis.}
\end{aligned}$$

Finally let us check that the chain (9) stabilizes after ω steps.

$$\begin{aligned}
&(c^* \circ \varphi)\bigl(\textstyle\bigwedge_{i\in\omega} (c^* \circ \varphi)^i \top_X\bigr) \\
&= (c^* \circ \varphi)\bigl(\textstyle\bigwedge_{i\in\omega} c_i^*(\varphi^i \top_1)\bigr) \quad \text{by the previous paragraph} \\
&= (c^* \circ \varphi)\bigl(c_\omega^*(\varphi^\omega \top_1)\bigr) \quad \text{by (B.7)} \\
&= c^*(Fc_\omega)^*\bigl(\varphi(\varphi^\omega \top_1)\bigr) \quad \text{by Def. 2.2} \\
&= c^*(Fc_\omega)^*\bigl(b^*(\varphi^\omega \top_1)\bigr) \quad \text{by Lem. 3.6} \\
&= c_\omega^*(\varphi^\omega \top_1) \quad \text{by } b \circ Fc_\omega \circ c = c_\omega\text{, see Step 1a)} \\
&= c_\omega^*(\textstyle\bigwedge_{i\in\omega} \pi_i^*(\varphi^i \top_1)) \quad \text{by (B.4)} \\
&= \textstyle\bigwedge_{i\in\omega} c_\omega^* \pi_i^*(\varphi^i \top_1) \\
&= \textstyle\bigwedge_{i\in\omega} c_i^*(\varphi^i \top_1) \\
&= \textstyle\bigwedge_{i\in\omega} (c^* \circ \varphi)^i \top_X \quad \text{by the previous paragraph.}
\end{aligned}$$

This concludes the proof. □

B.3 Proof of Prop. 3.11

Proof 1)

$$\begin{aligned}
f^*Q \le f^*d^*(\varphi Q) &\quad Q \text{ is an invariant} \\
= c^*(Ff)^*(\varphi Q) &\quad f \text{ is a homomorphism} \\
= (c^* \circ \varphi)(f^*Q) &\quad \text{by Def. 2.2.}
\end{aligned}$$

2) The coalgebras give rise to mediating arrows $X \overset{c_\omega}{\to} F^\omega 1$ and $Y \overset{d_\omega}{\to} F^\omega 1$, respectively, as in (7). It is easy to see that $c_\omega = d_\omega \circ f$ (using the universality of the limit $F^\omega 1$); using (8) the claim follows. □

B.4 Proof of Prop. 4.1

Proof It is easy to check each fiber $\mathbf{Coalg}(\varphi)_{X \overset{c}{\to} FX}$ is a poset. Let $(X \overset{c}{\to} FX) \overset{f}{\to} (Y \overset{d}{\to} FY)$ be an arrow in $\mathbf{Coalg}(F)$, and $P \overset{s}{\to} \varphi P$ be above $Y \overset{d}{\to} FY$. A Cartesian lifting of f are obtained as in the following diagram.

$$\begin{tikzcd}
\boxed{\mathbb{P}} & \varphi f^* P \arrow[r, "\varphi\overline{f}(P)"] & \varphi P \\
& f^* P \arrow[u, dashed, "t"] \arrow[r, "\overline{f}(P)"'] & P \arrow[u, "s"'] \\
\boxed{\mathbb{C}} & FX \arrow[r, "Ff"] & FY \\
& X \arrow[u, "c"] \arrow[r, "f"'] & Y \arrow[u, "d"']
\end{tikzcd}$$

Here we used the universality of the Cartesian lifting $\varphi\overline{f}(P)$ (see Def. 2.2).

The two forgetful functors constitute a map of fibrations: the commutativity (4) is obvious, and Cartesian liftings in $\begin{array}{c}\mathbf{Coalg}(\varphi)\\ \downarrow\overline{p}\\ \mathbf{Coalg}(F)\end{array}$ (which we constructed above) are based on the Cartesian liftings in $\begin{array}{c}\mathbb{P}\\ \downarrow p\\ \mathbb{C}\end{array}$. □

B.5 Proof of Prop. 4.2

Proof Given a φ-coalgebra $P \overset{s}{\to} \varphi P$ above $X \overset{c}{\to} FX$, we use the universality of the Cartesian lifting of c to obtain a $(c^* \circ \varphi)$-coalgebra as in the following diagram.

$$\begin{tikzcd}
c^* \varphi P \arrow[r, "\overline{c}(\varphi P)"] & \varphi P \\
P \arrow[u] \arrow[ur, "s"']
\end{tikzcd}$$

Conversely, given a $(c^* \circ \varphi)$-coalgebra $Q \overset{t}{\to} c^*(\varphi Q)$, we obtain a φ-coalgebra above $X \overset{c}{\to} FX$ as the following composite.

$$\begin{tikzcd}
c^* \varphi Q \arrow[r, "\overline{c}(\varphi Q)"] & \varphi Q \\
Q \arrow[u, "t"]
\end{tikzcd}$$

Then it is straightforward to see that the mappings are monotone and inverse to each other. The mappings commute with the forgetful functors since they do not change the carriers. □

B.6 Proof of Lem. 5.4

Proof The proof is by steps.

a) First we show that $\mathrm{Sub}(\mathbb{C})$ is complete and cocomplete. We rely on Lem. A.16.

We start with fiberwise limits in $\begin{array}{c}\mathrm{Sub}(\mathbb{C})\\ \downarrow\\ \mathbb{C}\end{array}$; the proof is like in [29, Example 1.8.3(iii)]. By Lem. A.4 an LFP category $\mathbb{C}$ is complete. This equips each fiber $\mathrm{Sub}(X)$ with arbitrary inf's $\bigwedge$ computed as wide pullbacks. A reindexing functor (by pullbacks) preserves these inf's since limits commute. Therefore by Lem. A.16 the total category $\mathrm{Sub}(\mathbb{C})$ is complete.

By the assumption that $\mathbb{C}$ is an LCCC, $\begin{array}{c}\mathrm{Sub}(\mathbb{C})\\ \downarrow\\ \mathbb{C}\end{array}$ has products $\prod_f \vdash f^*$ between fibers [29, Cor. 1.9.9].

Next we show that $\begin{array}{c}\mathrm{Sub}(\mathbb{C})\\ \downarrow\\ \mathbb{C}\end{array}$ has fiberwise colimits. Each fiber (which is a poset) has arbitrary inf's; hence it is a complete lattice and arbitrary sup's also exist. These sup's (i.e. colimits in a fiber) are preserved by reindexing f^* since the latter is a left adjoint $f^* \dashv \prod_f$.

We further show that $\begin{array}{c}\mathrm{Sub}(\mathbb{C})\\ \downarrow\\ \mathbb{C}\end{array}$ has coproducts $\coprod$ between fibers. An abstract proof can be given by Freyd's adjoint functor theorem (note that each fiber $\mathrm{Sub}(X)$ is a complete lattice, and that reindexing f^* preserves inf's). Instead we explicitly introduce $\coprod$ exploiting a factorization structure of LFP $\mathbb{C}$ (Lem. A.4.ii). Namely, given $(P \overset{m}{\rightarrowtail} X) \in \mathrm{Sub}(X)$ and $f : X \to Y$, the coproduct $\coprod_f P$ is defined by the $(\mathrm{StrongEpi}, \mathrm{Mono})$-factorization of $f \circ m$, as below.

$$\begin{tikzcd}
P \arrow[r, two heads] \arrow[d, tail, "m"'] & \coprod_f P \arrow[d, tail] \\
X \arrow[r, "f"'] & Y
\end{tikzcd} \tag{B.8}$$

The fact that $\coprod_f P \le Q$ if and only if $P \le f^* Q$ is easily proved using the diagonalization property of the factorization structure. This establishes $\coprod_f$ as a left adjoint to reindexing f^*. These coproducts $\coprod$ satisfy the Bech-Chevalley condition since the products $\prod$ do [29, Lem. 1.9.7]. Using Lem. A.16 we conclude that $\mathrm{Sub}(\mathbb{C})$ is cocomplete.

b) First we prove that, if P and X are both FP in $\mathbb{C}$, then $(P \overset{m}{\rightarrowtail} X)$ is FP in $\mathrm{Sub}(\mathbb{C})$. Let $(Q_I \overset{n_I}{\rightarrowtail} Y_I)_{I \in \mathbb{I}}$ be a filtered diagram in $\mathrm{Sub}(\mathbb{C})$; $(Q \overset{n}{\rightarrowtail} Y)$ its colimit; and $g : (P \overset{m}{\rightarrowtail} X) \to (Q \overset{n}{\rightarrowtail} Y)$ an arrow in $\mathrm{Sub}(\mathbb{C})$. By Lem. A.16 the colimit $(Q \rightarrowtail Y)$ can be explicitly described as

$$Y = \operatorname*{Colim}_{I \in \mathbb{I}} Y_I \ , \qquad Q = \bigvee_{I \in \mathbb{I}} \coprod\nolimits_{\kappa_I} Q_I \ , \tag{B.9}$$

where $(Y_I \overset{\kappa_I}{\to} Y)_{I \in \mathbb{I}}$ is a colimiting cocone.

Sublemma B.2 *The object $Q \in \mathbb{C}$ is a colimit $\mathrm{Colim}_{I \in \mathbb{I}} Q_I$ computed in $\mathbb{C}$.*

Proof (Of the sublemma) Both $(Q_I)_{I\in\mathbb{I}}$ and $(Y_I)_{I\in\mathbb{I}}$ are $\mathbb{I}$-shaped diagrams in $\mathbb{C}$ with a monotransformation $(Q_I \overset{n_I}{\rightarrowtail} Y_I)_I$. Therefore by [4, Cor. 1.60], the induced arrow $\mathrm{Colim}_I\, Q_I \to \mathrm{Colim}_I\, Y_I$ is monic, establishing $\mathrm{Colim}_I\, Q_I \in \mathrm{Sub}(Y)$. It suffices to find arrows a, b in the diagram below.

$$\begin{tikzcd}
\mathrm{Colim}_I\, Q_I \arrow[rr, shift left, "a"] \arrow[dr, tail] & & Q \arrow[ll, shift left, "b"] \arrow[dl, tail, "n"] \\
& Y &
\end{tikzcd} \tag{B.10}$$

The arrow a is obtained in the following way. Since κ_I is an arrow $(Q_I \rightarrowtail Y_I) \to (Q \rightarrowtail Y)$ in $\mathrm{Sub}(\mathbb{C})$, by (A.5) we have an arrow $Q_I \to Q$ in $\mathbb{C}$, for each $I \in \mathbb{I}$. These arrows induce a as a mediating arrow.

To obtain b in (B.10), since $Q = \bigvee_{I\in\mathbb{I}} \coprod_{\kappa_I} Q_I$ (see (B.9)), it suffices to find b_I below for each $I \in \mathbb{I}$.

$$\begin{tikzcd}
\coprod_{\kappa_I} Q_I \arrow[rr, "b_I"] \arrow[dr, tail] & & \mathrm{Colim}_I\, Q_I \arrow[dl, tail] \\
& Y &
\end{tikzcd}$$

This is obtained as the following diagonal fill-in. Recall that $Y = \mathrm{Colim}_I\, Y_I$.

$$\begin{tikzcd}
Q_I \arrow[r, two heads] \arrow[d] \arrow[rr, tail, bend left, "n_I"] & \coprod_{\kappa_I} Q_I \arrow[dl, dashed, "b_I"'] \arrow[d, tail] & Y_I \arrow[dl, "\kappa_I"] \\
\mathrm{Colim}_I\, Q_I \arrow[r, tail, "{[\kappa_I \circ n_I]_I}"'] & \mathrm{Colim}_I\, Y_I &
\end{tikzcd}$$

This proves Sublem. B.2. □

We are back in Step b). Since $g : (P \overset{m}{\rightarrowtail} X) \to (Q \overset{n}{\rightarrowtail} Y)$ is an arrow in $\mathrm{Sub}(\mathbb{C})$, we also have an arrow $g' : P \to Q$, such that $n \circ g' = g \circ m$, by (A.5). Now Q and Y are filtered colimits of $(Q_I)_{I\in\mathbb{I}}$ and $(Y_I)_{I\in\mathbb{I}}$, respectively (the former is by Sublem. B.2). Since P and X are FP, $I_0 \in \mathbb{I}$ can be chosen such that g factors through $Y_{I_0} \to Y$ and g' factors through $Q_{I_0} \to Q$. That is,

$$\begin{tikzcd}
P \arrow[r, "h'"] \arrow[d, tail, "m"'] \arrow[rr, bend left, "g'"] & Q_{I_0} \arrow[r] \arrow[d, tail, "n_{I_0}"'] & Q \arrow[d, tail, "n"] \\
X \arrow[r, "h"'] \arrow[rr, bend right, "g"'] & Y_{I_0} \arrow[r, "\kappa_{I_0}"'] & Y
\end{tikzcd} .$$

It is not (yet) necessarily the case that the square on the left commutes, i.e. $n_{I_0} \circ h' = h \circ m$. The two arrows give factorizations of the arrow $n \circ g' = g \circ m : P \to Y$ via $Y_{I_0} \overset{\kappa_{I_0}}{\to} Y$; since P is FP, there exists $I_1 \in \mathbb{I}$ with $i : I_0 \to I_1$ such that

$$(Yi) \circ n_{I_0} \circ h' = (Yi) \circ h \circ m$$

(essential uniqueness of factorization, [4, Def. 1.1]). It is clear that, for such I_1, the arrow g in $\mathrm{Sub}(\mathbb{C})$ factors through $(Q_{I_1} \rightarrowtail Y_{I_1}) \overset{\kappa_{I_1}}{\to} (Q \rightarrowtail Y)$. This concludes Step b) that $(P \rightarrowtail X)$ is FP in $\mathrm{Sub}(\mathbb{C})$.

c) Recall that

$$\mathbb{F}_{\mathrm{Sub}(\mathbb{C})} \;:=\; \{\, (P \rightarrowtail X) \mid P, X \in \mathbb{F} \} \;. \tag{B.11}$$

The set $\mathbb{F}_{\mathrm{Sub}(\mathbb{C})}$ in (B.11) is small, since $\mathbb{F}$ is small and $\mathbb{C}$ is well-powered [4, Rem. 1.56].

d) In the remainder of the proof we show that every object $(Q \overset{n}{\rightarrowtail} Y) \in \mathrm{Sub}(\mathbb{C})$ is a colimit of the canonical diagram with respect to $\mathbb{F}_{\mathrm{Sub}(\mathbb{C})}$ from (B.11). Let $(Q_J \overset{n_J}{\rightarrowtail} Y_J)_{J\in\mathbb{J}}$ be the canonical diagram (i.e. $\mathbb{J} = (\mathbb{F}_{\mathrm{Sub}(\mathbb{C})}{\downarrow}n)$), with the canonical cocone

$$\Big(\, (Q_J \overset{n_J}{\rightarrowtail} Y_J) \xrightarrow{f_J} (Q \overset{n}{\rightarrowtail} Y)\, \Big)_{J\in\mathbb{J}} \ . \tag{B.12}$$

Let us denote the canonical diagram for $Y \in \mathbb{C}$ with respect to $\mathbb{F}$ by $(Y'_I)_{I\in\mathbb{I}}$ (i.e. $\mathbb{I} = (\mathbb{F}{\downarrow}Y)$), with a canonical cocone $(Y'_I \overset{\kappa_I}{\to} Y)_{I\in\mathbb{I}}$. The cocone is colimiting ($Y = \mathrm{Colim}_{I\in\mathbb{I}}\, Y'_I$) since $\mathbb{C}$ is LFP. In this Step d) we show $\mathrm{Colim}_{J\in\mathbb{J}}\, Y_J \cong \mathrm{Colim}_{I\in\mathbb{I}}\, Y'_I = Y$. A cocone $(Y_J \overset{\kappa_{I_J}}{\to} \mathrm{Colim}_{I\in\mathbb{I}}\, Y'_I)_{J\in\mathbb{J}}$ can be defined by finding (unique) $I_J \in \mathbb{I}$ such that $Y'_{I_J} \overset{f'_{I_J}}{\to} Y$ is equal to $Y_J \overset{f_J}{\to} Y$. In order to see that this cocone is colimiting, let $(Y_J \overset{g_J}{\to} Z)_{J\in\mathbb{J}}$ be another cocone (recall that $\mathbb{J} = (\mathbb{F}_{\mathrm{Sub}(\mathbb{C})}{\downarrow}n)$).

Sublemma B.3 *If the indices $J, J' \in \mathbb{J}$ satisfy $Y_J = Y_{J'}$ and $f_J = f_{J'}$ (cf. (B.12)), then $g_J = g_{J'}$.*

Proof (Of the sublemma) Let 0 be an initial object in $\mathbb{C}$. We first prove that 0 is a subobject of any object of $\mathbb{C}$. Indeed, since $\mathbb{C}$ is an LCCC (hence a CCC), its initial object 0 is *strict*, meaning that if $U \to 0$ exists then U is also initial (a standard result; see e.g. [33, Lem. 1.5.12]). This makes any arrow $0 \to V$ in $\mathbb{C}$ a mono.

Therefore we have an object $(0 \rightarrowtail Y_J)$ in $\mathrm{Sub}(\mathbb{C})$ induced by initiality. In view of (A.5), an arrow f_J in $\mathbb{C}$ induces an arrow

$$(0 \rightarrowtail Y_J) \xrightarrow{f_J = f_{J'}} (Q \overset{n}{\rightarrowtail} Y) \quad \text{in } \mathrm{Sub}(\mathbb{C});$$

therefore there exists an index $J'' \in \mathbb{J} = (\mathbb{F}_{\mathrm{Sub}(\mathbb{C})}{\downarrow}n)$ such that $(0 \rightarrowtail Y_J) = (Q_{J''} \overset{n_{J''}}{\rightarrowtail} Y_{J''})$, and $f_{J''} = f_J = f_{J'}$.

This gives us the following diagram in $\mathbb{J} = (\mathbb{F}_{\mathrm{Sub}(\mathbb{C})}{\downarrow}n)$.

$$\begin{tikzcd}
(Q_J \overset{n_J}{\rightarrowtail} Y_J) & (Q_{J''} \overset{n_{J''}}{\rightarrowtail} Y_{J''}) \arrow[l, "\mathrm{id}_{Y_J}"'] \arrow[r, "\mathrm{id}_{Y_{J'}}"] & (Q_{J'} \overset{n_{J'}}{\rightarrowtail} Y_{J'})
\end{tikzcd}$$

Therefore, since $(Y_J \overset{g_J}{\to} Z)_{J\in\mathbb{J}}$ is a cocone, the following diagram in $\mathbb{C}$ must commute.

$$\begin{tikzcd}
Y_J \arrow[dr, bend right, "g_J"'] & Y_{J''} \arrow[l, "\mathrm{id}"'] \arrow[r, "\mathrm{id}"] \arrow[d, "g_{J''}"'] & Y_{J'} \arrow[dl, bend left, "g_{J'}"] \\
& Z &
\end{tikzcd}$$

This proves $g_J = g_{J'} = g_{J''}$, as required in Sublem. B.3. □

We are back in Step d). For each $Y'_I \overset{f'_I}{\to} Y$ in $(\mathbb{F}{\downarrow}Y)$, there exists $J_I \in \mathbb{J}$ such that $Y_{J_I} = Y'_I$ and $f_{J_I} = f'_I$ (one can take the initial object 0, which is FP, as Q_{J_I}). Using such J_I we obtain an arrow

$$[g_{J_I}]_{I\in\mathbb{I}} \ : \ \operatorname*{Colim}_{I\in\mathbb{I}} Y'_I \longrightarrow Z \ .$$

That this is a mediating arrow, i.e. that the diagram

$$\begin{tikzcd}
Y_J \arrow[rr, "g_J"] \arrow[dr, "\kappa_{I_J}"'] & & Z \\
& \mathrm{Colim}_{I\in\mathbb{I}}\, Y'_I \arrow[ur, "{[g_{J_I}]_{I\in\mathbb{I}}}"'] &
\end{tikzcd}$$

commutes, is precisely the content of Sublem. B.3. Uniqueness of a mediating arrow is easy, too. This proves $\mathrm{Colim}_{J\in\mathbb{J}} Y_J \cong \mathrm{Colim}_{I\in\mathbb{I}} Y'_I$.

e) By Step d) we obtain $Y = \mathrm{Colim}_{J\in\mathbb{J}} Y_J$. In view of Lem. A.16, we are done if we show that $Q = \bigvee_{J\in\mathbb{J}} \coprod_{f_J} Q_J$.

One direction $Q \geq \bigvee_{J\in\mathbb{J}} \coprod_{f_J} Q_J$ is easy: since f_J in (B.12) is an arrow in $\mathrm{Sub}(\mathbb{C})$ we have $Q_J \leq f_J^* Q$, that is, $\coprod_{f_J} Q_J \leq Q$, for each $J \in \mathbb{J}$.

To prove the other direction ($Q \leq \bigvee_{J\in\mathbb{J}} \coprod_{f_J} Q_J$), let $(Q_K)_{K\in\mathbb{K}}$ be the canonical diagram for Q with respect to $\mathbb{F}$ (i.e. $\mathbb{K} = (\mathbb{F} \downarrow Q)$), with the canonical cocone $(Q_K \overset{c_K}{\to} Q)_{K\in\mathbb{K}}$. Then $Q = \mathrm{Colim}_{K\in\mathbb{K}} Q_K$ since $\mathbb{C}$ is LFP; furthermore, much like the proof of Sublem. B.2, we can show that $\mathrm{Colim}_{K\in\mathbb{K}} Q_K \cong \bigvee_{K\in\mathbb{K}} \coprod_{c_K} Q_K$. Hence it suffices to show

$$\textstyle\coprod_{c_K} Q_K \leq \bigvee_{J\in\mathbb{J}} \coprod_{f_J} Q_J \quad \text{in } \mathrm{Sub}(Y)\text{, for each } K \in \mathbb{K}. \tag{B.13}$$

It is easy to see (using (A.5)) that $n \circ c_K$ is an arrow

$$(Q_K \overset{\mathrm{id}}{\rightarrowtail} Q_K) \xrightarrow{n\circ c_K} (Q \overset{n}{\rightarrowtail} Y)$$

in $\mathrm{Sub}(\mathbb{C})$. Since $Q_K \in \mathbb{F}$, this arrow $n \circ c_K$ is an object of the index category $\mathbb{J} = (\mathbb{F}_{\mathrm{Sub}(\mathbb{C})} \downarrow n)$. This yields

$$\textstyle\coprod_{n\circ c_K} Q_K \leq \bigvee_{J\in\mathbb{J}} \coprod_{f_J} Q_J \enspace . \tag{B.14}$$

Now the following diagram shows that $\coprod_{n\circ c_K} Q_K = \coprod_{c_K} Q_K$ as a subobject of Y, via the uniqueness of factorization.

$$\begin{tikzcd}
Q_K \arrow[r, two heads] \arrow[rr, bend left, "c_K"] \arrow[dr, two heads] & \coprod_{c_K} Q_K \arrow[r, tail] & Q \arrow[r, tail, "n"] & Y \\
& \coprod_{n\circ c_K} Q_K \arrow[u, "\cong"] \arrow[urr, tail] & &
\end{tikzcd}$$

Therefore (B.14) proves (B.13). This concludes the proof. □

B.7 Proof of Lem. 5.2

Proof The only nontrivial part is the ⇐ direction of Cond. iii). For that it suffices to show that arbitrary $P \in \mathbb{P}$ is a colimit of the diagram $(\kappa_I^* P)_{I\in\mathbb{I}}$. Here $\mathbb{I}$ and κ_I are as in Cond. iii).

By Lem. A.16 the colimit $\mathrm{Colim}_{I\in\mathbb{I}} \kappa_I^* P$ is described as $\bigvee_{I\in\mathbb{I}} \coprod_{\kappa_I} \kappa_I^* P$ using a sup $\bigvee$ in $\mathbb{P}_X$, since $(X_I \overset{\kappa_I}{\to} X)_{I\in\mathbb{I}}$ is colimiting. We have $\coprod_{\kappa_I} \kappa_I^* P \leq P$ as a counit of an adjunction; therefore $\mathrm{Colim}_{I\in\mathbb{I}} \kappa_I^* P \leq P$.

Thus it suffices to show that $P \leq \mathrm{Colim}_{I\in\mathbb{I}} \kappa_I^* P$ in $\mathbb{P}_X$. Let $(P_J)_{J\in\mathbb{J}}$ be a diagram in $\mathbb{P}$ such that $P_J \in \mathbb{F}_{\mathbb{P}}$ and there is a colimiting cocone $(P_J \overset{g_J}{\to} P)_{J\in\mathbb{J}}$. Such a diagram exists since $\mathbb{F}_{\mathbb{P}}$ is dense.

By the assumption, for each J the object $P_J \in \mathbb{F}_{\mathbb{P}}$ lies above an object in $\mathbb{F}_{\mathbb{C}}$. Therefore the arrow $pg_J : pP_J \to pP = X$ is an object of $(\mathbb{F}_{\mathbb{C}} \downarrow X)$; since $\mathbb{I} = (\mathbb{F}_{\mathbb{C}} \downarrow X)$, we can choose $I_J \in \mathbb{I}$ such that $\kappa_{I_J} = pg_J$. Now an arrow $P_J \overset{g_J}{\to} P$ in $\mathbb{P}$ induces

$$P_J \leq (pg_J)^* P = \kappa_{I_J}^* P \tag{B.15}$$

by the universality of Cartesian arrows. We proceed as follows.

$$\begin{aligned} P = \mathrm{Colim}_{J\in\mathbb{J}}\, P_J \overset{(*)}{=} \textstyle\bigvee_{J\in\mathbb{J}} \coprod_{pg_J} P_J \overset{(\dagger)}{\leq} \bigvee_{J\in\mathbb{J}} \coprod_{\kappa_{I_J}} \kappa^*_{I_J} P & \\ \leq \bigvee_{I\in\mathbb{I}} \textstyle\coprod_{\kappa_I} \kappa^*_I P \overset{(*)}{=} \mathrm{Colim}_{I\in\mathbb{I}}\, \kappa^*_I P \ . & \end{aligned}$$

For $(*)$ we used Lem. A.16; $(\dagger)$ holds since I_J is chosen so that $\kappa_{I_J} = pg_J$ and (B.15) hold. This concludes the proof. □

B.8 Proof of Lem. 5.6

Proof 1a) Let us first see that $\mathrm{Fam}(\Omega)$ is cocomplete. In view of Lem. A.16, it suffices to show that $\begin{array}{c}\mathrm{Fam}(\Omega)\\ \downarrow\\ \mathbf{Sets}\end{array}$ has fiberwise colimits and coproducts $\coprod$ between fibers (the base category **Sets** is cocomplete). The former follows from Ω being a complete lattice; the latter is shown from [29, Lem. 1.9.5].

1b) Before going on we prove the following.

Sublemma B.4 *An arrow in* $\mathrm{Fam}(\Omega)$ *is a mono if and only if its underlying function is a mono in* **Sets**.

Proof (Of Sublem. B.4) The 'if' part is obvious. For the 'only if' part, let $(X \xrightarrow{f} \Omega) \overset{m}{\rightarrowtail} (Y \xrightarrow{g} \Omega)$ be a monic arrow in $\mathrm{Fam}(\Omega)$, and $k, l : U \to X$ be arrows in **Sets** such that $m \circ k = m \circ l$. This induces the following situation in $\mathrm{Fam}(\Omega)$:

$$\begin{tikzcd} (\bot : U \to \Omega) \arrow[r, shift left, "k"] \arrow[r, shift right, "l"'] & (f : X \to \Omega) \arrow[r, tail, "m"] & (g : Y \to \Omega) \end{tikzcd} \quad ,$$

where $\bot : U \to \Omega$ is the constant function to the least element $\bot \in \Omega$. Therefore $k = l$; this proves Sublem. B.4. □

1c) We prove that each $(X \xrightarrow{f} \Omega) \in \mathbb{F}_{\mathrm{Fam}(\Omega)}$ is FG (Def. A.7) in $\mathrm{Fam}(\Omega)$. Let $\big((Y_I \xrightarrow{g_I} \Omega) \xrightarrow{h_I} (Y \xrightarrow{g} \Omega)\big)_{I\in\mathbb{I}}$ be a colimiting cocone from a directed diagram $\mathbb{I}$ whose arrows are all monos; and $(X \xrightarrow{f} \Omega) \xrightarrow{k} (Y \xrightarrow{g} \Omega)$ be an arrow in $\mathrm{Fam}(\Omega)$. We aim at showing that k factors through some h_I.

By Lem. A.16 we obtain that $Y = \mathrm{Colim}_{I\in\mathbb{I}} Y_I$; and that

$$\begin{aligned} g(y) &= (\textstyle\bigvee_{I\in\mathbb{I}} \coprod_{h_I} g_I)(y) = \bigvee_{I\in\mathbb{I}}\big((\coprod_{h_I} g_I)(y)\big) \\ &= \textstyle\bigvee_{I\in\mathbb{I}}\big(\bigvee_{y'\in h_I^{-1}(y)} g_I(y')\big) \quad \text{for each } y \in Y\,. \end{aligned} \tag{B.16}$$

The first equality is by Lem. A.16; the second is because the order in the fiber $\mathrm{Fam}(\Omega)_Y = \Omega^Y$ is pointwise; and the third is by the concrete description [29, Lem. 1.9.5] of $\coprod$ in $\begin{array}{c}\mathrm{Fam}(\Omega)\\ \downarrow\\ \mathbf{Sets}\end{array}$.

We observe that each h_I is a mono in $\mathrm{Fam}(\Omega)$. To see it, $(Y_I \xrightarrow{h_I} Y)_{I\in\mathbb{I}}$ is a colimiting cocone in **Sets** from a directed diagram of monos; since **Sets** is LFP, we can use [4, Prop. 1.62]; and then we use Sublem. B.4.

Now we have

$$f(x) \le g(k(x)) = \bigvee_{I\in\mathbb{I}}\bigl(\bigvee_{y'\in h_I^{-1}(k(x))} g_I(y')\bigr) \ ;$$

here the first inequality is because k is an $\mathrm{Fam}(\Omega)$-arrow; and the second equality is from (B.16). By the assumptions that $f(x)$ is compact and that X is finite, there exists $I_0 \in \mathbb{I}$ such that $f(x) \le \bigvee_{y'\in h_{I_0}^{-1}(k(x))} g_{I_0}(y')$ for each $x \in X$ (recall that $\mathbb{I}$ is filtered). Furthermore, since $X \overset{k}{\to} Y = \mathrm{Colim}_{I\in\mathbb{I}} Y_I$ is an arrow from an FP object (in **Sets**) to a directed colimit, it factors through some h_{I_1}:

$$\begin{tikzcd}
X \arrow[r, "l_{I_1}"] \arrow[rd, "k"'] & Y_{I_1} \arrow[d, "h_{I_1}"] \\
& Y
\end{tikzcd} \quad .$$

By choosing I_2 such that $I_0, I_1 \le I_2$, we have

$$f(x) \le \bigvee_{y'\in h_{I_2}^{-1}(k(x))} g_{I_2}(y') = g_{I_2}(l_{I_2}(x)) \quad \text{for each } x \in X;$$

here the last equality holds since h_I is an injection and $h_I(l_I(x)) = k(x)$. This proves that l_{I_2} is a $\mathrm{Fam}(\Omega)$-arrow $(X \overset{f}{\to} \Omega) \to (Y_{I_2} \overset{g_{I_2}}{\to} \Omega)$, hence $k = h_{I_2} \circ l_{I_2}$ in $\mathrm{Fam}(\Omega)$. This concludes Step 1c).

1d) The collection $\mathbb{F}_{\mathrm{Fam}(\Omega)}$ is obviously small.

1e) We are done if we prove that every object $P \in \mathrm{Fam}(\Omega)$ is a directed colimit of its subobjects from $\mathbb{F}_{\mathrm{Fam}(\Omega)}$. This easily follows from the fact that the same is true in **Sets** (obvious) and in Ω (being an algebraic lattice). □

B.9 Proof of Lem. 5.9

Proof Any presheaf $P \in \mathbf{Sets}^{\mathbb{A}}$ has a canonical isomorphism $\mathrm{Colim}_{(A,p)\in\int P}\, \mathbf{y}A \cong P$ induced by $(\mathbf{y}A)(B) = \mathbb{A}(A,B) \ni g \mapsto P(g)(p) \in P(A)$ for $A \in \mathbb{A}$ and $p \in P(A)$, where $\int P$ is the category of elements of P. (Remark: The category of elements of covariant functor $P\colon \mathbb{A} \to \mathbf{Sets}$ consists of objects (A,p) in the above and arrows $h\colon (A,p) \to (B,q)$ for all arrows $h\colon B \to A$ in $\mathbb{A}$ such that $P(h)(q) = p$.) In the situation, we assume that P is a subpresheaf of $\mathbf{y}X$. Then $P(g) = (\mathbf{y}X)(g) = (g \circ \_)$ shows that arrows $\{(\_ \circ f) = \mathbf{y}f\colon \mathbf{y}A \to \mathbf{y}X\}_{(A,f)}$ induce the composition $(\mathrm{Colim}_{(A,f)\in\int P}\, \mathbf{y}A) \cong P \hookrightarrow \mathbf{y}X$. Regarding P as the image $\mathrm{Im}\bigl((\mathrm{Colim}_{(A,f)\in\int P}\, \mathbf{y}A) \rightarrowtail \mathbf{y}X\bigr)$, the following component-wise calculation on objects $B \in \mathbb{A}$ shows $P = \bigcup_{(A,f)\in\int P} \operatorname{Im} \mathbf{y}f$:

$$
\begin{aligned}
&\big(\operatorname{Im}\big((\operatorname*{Colim}_{(A,f)\in\int P} \mathbf{y}A) \to \mathbf{y}X\big)\big)(B) \\
&\underset{(*)}{=} \operatorname{Im}\big((\operatorname*{Colim}_{(A,f)\in\int P} \mathbf{y}A)(B) \to \mathbf{y}X(B)\big) \\
&\underset{(\dagger)}{=} \operatorname{Im}\Big(\big(\operatorname*{Colim}_{(A,f)\in\int P} \big(\mathbf{y}A(B)\big)\big) \to \mathbf{y}X(B)\Big) \\
&= \operatorname{Im}\Big(\big(\textstyle\coprod_{(A,f)\in\int P} \big(\mathbf{y}A(B)\big)\big)/{\sim} \to \mathbf{y}X(A)\Big) \\
&= \operatorname{Im}\Big(\big(\textstyle\coprod \big(\mathbf{y}A(B)\big)\big) \twoheadrightarrow \big(\textstyle\coprod \big(\mathbf{y}A(B)\big)\big)/{\sim} \to \mathbf{y}X(A)\Big) \\
&= \textstyle\bigcup_{(A,f)\in\int P} \operatorname{Im}\big(\mathbf{y}A(B) \to \mathbf{y}X(B)\big) \\
&\underset{(*)}{=} \textstyle\bigcup_{(A,f)\in\int P} \big((\operatorname{Im} \mathbf{y}f)(B)\big) \underset{(\dagger)}{=} \big(\textstyle\bigcup_{(A,f)\in\int P} \operatorname{Im} \mathbf{y}f\big)(B),
\end{aligned}
$$

where $\sim$ is a suitable equivalence relation in the explicit formula of colimits in **Sets**. Note that Im in the first line and the last line are the images in $\mathbf{Sets}^{\mathbb{A}}$ while they denote the images in **Sets** elsewhere; and that $(*)$ and $(\dagger)$ holds because limits and colimits are component-wise, with a fact for $(*)$ that an image is an equalizer of a cokernel pair in both **Sets** and $\mathbf{Sets}^{\mathbb{A}}$. Therefore, there are only finitely many subpresheaves P of $\mathbf{y}X$ if $\{\operatorname{Im} \mathbf{y}f \mid A \in \mathbb{A}, f\colon X \to A\}$ is finite.

For the special case in the second half, we first prove the following.

Sublemma B.5 *The inclusion relation on* $\{\operatorname{Im} \mathbf{y}f \mid A \in \mathbb{A}, f\colon X \to A\}$ *is derived from a preorder* $\lesssim$ *on* $\{f \mid A \in \mathbb{A}, f\colon X \to A\}$ *such that* $(f\colon X \to A) \lesssim (g\colon X \to B)$ *iff* $f = h \circ g$ *for some* $h\colon B \to A$.

Proof (Of Sublem. B.5) If $\operatorname{Im} \mathbf{y}f \subseteq \operatorname{Im} \mathbf{y}g$, then $f = (\mathbf{y}f)_A(\mathrm{id}_A) \in \operatorname{Im}(\mathbf{y}f)(A) \subseteq \operatorname{Im}(\mathbf{y}g)(A) = \{h \circ g \mid h\colon B \to A\}$. Conversely, for $f = h \circ g$, any arrow $k \circ f = (\mathbf{y}f)_C(k) \in \operatorname{Im}(\mathbf{y}f)(C)$ is in $\operatorname{Im}(\mathbf{y}g)(C)$ because $k \circ f = k \circ h \circ g = (\mathbf{y}g)_C(k \circ h)$. □

It is enough to show that $\operatorname{Quot}(X) \ni Y \mapsto \operatorname{Im}(\mathbf{y}Y \rightarrowtail \mathbf{y}X) \in \{\operatorname{Im} \mathbf{y}f \mid A \in \mathbb{A}, f\colon A \to X\}$ is a bijection. It is obviously injective because epis $e\colon X \twoheadrightarrow Y$ and $e'\colon X \twoheadrightarrow Y'$ factor through each other if and only if e and e' are the same objects in $\operatorname{Quot}(X)$. We shall prove the mapping is surjective. Let $f\colon X \to A$ be an arbitrary arrow and $f = m \circ e$ be its factorization. Then, $\operatorname{Im} \mathbf{y}f \subseteq \operatorname{Im} \mathbf{y}e$ and conversely, we also have $\operatorname{Im} \mathbf{y}e \subseteq \operatorname{Im} \mathbf{y}f$ by $e = r \circ f$ for a retraction r of m. Therefore, $\operatorname{Im} \mathbf{y}f = \operatorname{Im} \mathbf{y}e$ is a image of the mapping. □

B.10 Proof of Cor. 5.10

Sublemma B.6 *Let* $(X_I)_I$ *be a finite diagram in* $\mathbf{Sets}^{\mathbb{A}}$. *If* $\operatorname{Sub}(X_I)$ *is finite for each* I, *then so is* $\operatorname{Sub}(\operatorname{Colim}_I X_I)$.

Proof (Of Sublem. B.6) In a topos (hence a regular category) $\mathbf{Sets}^{\mathbb{A}}$ coproducts are disjoint (see e.g. [29]); thus we have

$$\operatorname{Sub}(X_1 + \cdots + X_n) \;\cong\; \operatorname{Sub}(X_1) \times \cdots \times \operatorname{Sub}(X_n) \;\; .$$

Let $X \rightrightarrows Y \overset{e}{\twoheadrightarrow} Z$ be a coequalizer in $\mathbf{Sets}^{\mathbb{A}}$. The correspondence $e^* : \mathrm{Sub}(Z) \to \mathrm{Sub}(Y)$ is easily seen to be injective. Indeed, assume $P \ncong P'$ in $\mathrm{Sub}(Z)$; then $PA \ncong P'A$ for some $A \in \mathbb{A}$ in $\mathbf{Sets}$, and since e_A is surjective, we have

$$(e^*P)A = e_A^{-1}(PA) \ \ncong\ e_A^{-1}(P'A) = (e^*P)A \ \ .$$

Therefore if $\mathrm{Sub}(Y)$ is finite, so is $\mathrm{Sub}(Z)$. This concludes the proof of the sublemma. □

Proof (Of Cor. 5.10) By Example 5.8, Lem. 5.1, Sublem. B.6, and Cor. 5.5. □

Quasicontinuous Domains and the Smyth Powerdomain

Reinhold Heckmann[2]

AbsInt Angewandte Informatik GmbH
Science Park 1
D-66123 Saarbrücken, Germany

Klaus Keimel[1,3]

Fachbereich Mathematik
Technische Universität Darmstadt
D-64289 Darmstadt, Germany

Abstract

In Domain Theory quasicontinuous domains pop up from time to time generalizing slightly the powerful notion of a continuous domain. It is the aim of this paper to show that quasicontinuous domains occur in a natural way in relation to the powerdomains of finitely generated and compact saturated subsets. Properties of quasicontinuous domains seem to be best understood from that point of view. This is in contrast to the previous approaches where the properties of a quasicontinuous domain were compared primarily with the properties of the lattice of Scott-open subsets. We present a characterization of those domains that occur as domains of nonempty compact saturated subsets of a quasicontinuous domain.
A set theoretical lemma due to M. E. Rudin has played a crucial role in the development of quasicontinuous domains. We present a topological variant of Rudin's Lemma where irreducible sets replace directed sets. The notion of irreducibility here is that of a nonempty set that cannot be covered by two closed sets except if already one of the sets is covering it. Since directed sets are the irreducible sets for the Alexandroff topology on a partially ordered set, this is a natural generalization. It allows a remarkable characterization of sober spaces.
For this we denote by $\mathfrak{Q}X$ the space of nonempty compact saturated subsets (with the upper Vietoris topology) of a topological space X. The following properties are equivalent: (1) X is sober, (2) $\mathfrak{Q}X$ is sober, (3) X is *strongly* well-filtered in the following sense: Whenever $\mathcal{A}$ is an irreducible subset of $\mathfrak{Q}X$ and U an open subset of X such that $\bigcap \mathcal{A} \subseteq U$, then $K \subseteq U$ for some $K \in \mathcal{A}$. This result fills a gap in the existing literature.

Keywords: Quasicontinuous Domains, M. E. Rudin's Lemma, Powerdomains of compact saturated subsets.

[1] During working on this paper, the second author profited from a research visit to Nanyang Technological University supported by Academic Research Fund No. RP/10 HKW. Particular thanks to Dr. Ho Weng Kin and Dr. Zhao Dongsheng.

[2] Email: heckmann@absint.com

[3] Email: keimel@mathematik.tu-darmstadt.de

1 Introduction

In this paper we deal with the powerspace of compact saturated sets, quasicontinuous domains and variants of Rudin's Lemma. We intend to show that these three ingredients are inseparably tied together.

Quasicontinuous domains introduced by Gierz, Lawson and Stralka [5] capture many of the essential features of continuous domains. Recently they have attracted increased attention through the remarkable work of J. Goubault-Larrecq [6] and through a paper by Li and Xu [11].

An important result concerning continuous domains is their characterization by properties of their Scott topology. A dcpo is continuous if and only if its lattice of Scott open subsets is completely distributive. Gierz, Lawson and Stralka [5] have characterized quasicontinuous domains by the property that their lattice of Scott-open subsets is hypercontinuous. One of the characterizations of hypercontinuous lattices is that they are images of completely distributive lattices under maps preserving arbitrary meets and directed joins.

A characterization of the lattice of open subsets is equivalent to a characterization of the opposite lattice of closed subsets. The lattice of Scott-closed subsets of a dcpo is often called the Hoare or lower powerdomain of a dcpo. Thus, one can say that Gierz, Lawson and Stralka have characterized quasicontinuous domains through their lower powerdomains.

In this paper we intend to show that quasicontinuous domains should be tied up with the Smyth or upper powerdomain [15,16] rather than the lower powerdomain. We show that among dcpos the quasicontinuous domains can be characterized by the property that the poset of finitely generated upper sets ordered by reverse inclusion is a continuous poset. We claim that this opens useful insights and simpler proofs for known properties (see 4.5). We finish with a characterization of quasicontinuous domains through properties of their upper powerdomains (see Theorem 4.9).

From the beginning, the development of the notion of a quasicontinuous domain was dependent on a set theoretical lemma. In fact, M. E. Rudin provided the appropriate lemma as an answer to a question asked by Gierz, Lawson and Stralka, when they prepared the paper [5], where the notion of a quasicontinuous domain was introduced. In the same spirit, variants of Rudin's Lemma are the third ingredient of this paper (see Section 3). Rudin's original lemma is captured in Lemma 3.4 and Corollary 3.5. We also need it in our approach to quasicontinuous domains in Lemma 4.1.

A new topological variant of Rudin's Lemma is presented in Lemma 3.1; directed sets in Rudin's original Lemma are viewed as special cases of irreducible sets in topological spaces. This lemma allows a characterization of sober spaces (see Theorem 3.13). We use this theorem for a simplified proof of the sobriety of quasicontinuous posets (see Property 4.5(vii)).

Theorem 3.13 solves an open problem. A topological space had been called well-filtered if, whenever $\bigcap \mathcal{F} \subseteq U$ for a filter basis $\mathcal{F}$ of compact saturated sets and an open subset U, then $K \subseteq U$ for some $K \in \mathcal{F}$. It is known that every sober space is well-filtered. Conversely every locally compact well-filtered space is sober (Theorem [4, II-1.21]). But sobriety is not characterized by well-filterednes in general (for a counterexample see [10]). Theorem 3.13 tells us that sobriety is characterized by the property of being strongly well-filtered. By this we mean that, whenever $\mathcal{A}$ is an irreducible set in the hyperspace of compact saturated subsets (with the upper Vietoris topology) such that $\bigcap \mathcal{A}$ is contained in an open set U, then $K \subseteq U$ for some $K \in \mathcal{A}$.

2 Preliminaries

2.1 Order theoretical notions

For a partially ordered set (= poset) P, more generally for a preordered set, we fix the following terminology:

$D \subseteq P$ is *directed* if D is nonempty and if for any d_1, d_2 in D there is a d in D above d_1 and d_2.

In a poset P, a directed subset D may or may not have a least upper bound. We adopt the following convention: if we write $\bigvee^{\uparrow} D$ then we mean that D is a directed subset of P which has a least upper bound in P which we denote by $\bigvee^{\uparrow} D$.

P is *directed complete* (a *dcpo*) if every directed subset D of P has a least upper bound $\bigvee^{\uparrow} D$.

For $a \in P$ let $\uparrow a$ denote the set of all $x \in P$ with $a \leq x$ and, for a subset A, let $\uparrow A = \bigcup_{a \in A} \uparrow a$. A subset A of P is an *upper set* if $A = \uparrow A$. We denote by $\mathfrak{U}X$ the collection of all upper sets in X. The order dual concepts are $\downarrow a$, $\downarrow A$ and *lower set*.

For any set X, we denote by $\mathfrak{P}X$ the set of all subsets and by $\mathfrak{P}_f X$ the collection of all finite subsets; the letters F, G, H will always denote nonempty finite subsets.

If X is a partially ordered set, more generally a preordered set, we introduce a preorder $\sqsubseteq$ on the powerset $\mathfrak{P}X$, sometimes called the *Smyth preorder*, by

$$A \sqsubseteq B \iff \uparrow B \subseteq \uparrow A,$$

that is, $A \sqsubseteq B$ iff for every element $b \in B$ there is an element $a \in A$ with $a \leq b$. On the collection $\mathfrak{U}X$ of upper sets, $\sqsubseteq$ is a partial order, namely reverse inclusion.

We denote by

$$\eta_X\colon X \to \mathfrak{P}X \quad \text{the map} \quad \eta_X(x) = {\uparrow}x$$

which is an order embedding.

Every topological space X carries a natural (pre-)order, the *specialization (pre-)order* $x \leq y$ iff $x \in \mathsf{cl}\{y\}$, the closure of the singleton $\{y\}$. The previous order theoretical concepts can be applied to the specialization (pre-)order. And when we apply order theoretical notions to topological spaces, they always refer to the specialization (pre-)order. A subset of a topological space that is an upper set for its specialization (pre-)order is also called a *saturated* set.

Conversely, every poset X can be topologized in various ways. The upper sets form the *Alexandroff topology* $\mathfrak{U}X$. A coarser topology is the *Scott topology* σX: A subset $U \subseteq X$ is *Scott-open* if U is an upper set and if $\bigvee^{\uparrow} D \in U \Rightarrow D \cap U \neq \emptyset$, that is, if for every directed set D with $\bigvee^{\uparrow} D \in U$, there is a $d \in D$ with $d \in U$, provided that D has a least upper bound in X. The Scott-open sets form indeed a topology.

2.2 Compact and supercompact sets

A subset K of a topological space X is *compact* if for all directed families $(U_i)_{i\in I}$ of opens, $K \subseteq \bigcup_{i\in I} U_i$ implies $K \subseteq U_k$ for some k in I. It is *supercompact* if for arbitrary families $(U_i)_{i\in I}$ of opens, $K \subseteq \bigcup_{i\in I} U_i$ implies $K \subseteq U_k$ for some k in I.

Using that $K \subseteq U$ if and only if K does not meet $C = X \setminus U$, compactness can also be characterized using closed instead of open sets:

Fact 2.1 *A set K is compact iff for all filtered families $(C_i)_{i\in I}$ of closed sets, K meets $\bigcap_{i\in I} C_i$ whenever K meets all C_i. A set K is supercompact iff for all families $(C_i)_{i\in I}$ of closed sets, K meets $\bigcap_{i\in I} C_i$ whenever K meets all C_i.*

Note that a subset K is compact if and only if its *saturation*, the upper set ${\uparrow}K$ generated by K w.r.t. the specialization (pre)-order, is compact.

Fact 2.2 *The supercompact saturated sets of a topological space X are exactly the sets ${\uparrow}x$ with x in X.*

Proof. The sets ${\uparrow}x$ are clearly supercompact and saturated. For the opposite direction, let S be a supercompact upper set. The set S meets all sets of the family $({\downarrow}a)_{a\in S}$ of closed sets. By supercompactness, it meets $\bigcap_{a\in S} {\downarrow}a$. Let x be a member of $S \cap \bigcap_{a\in S} {\downarrow}a$. Since S is an upper set, ${\uparrow}x \subseteq S$ holds. On the other hand, x is in ${\downarrow}a$ for all a in S, whence $S \subseteq {\uparrow}x$. □

2.3 The Upper Powerspace

On the powerset $\mathfrak{P}X$ of all subsets of a topological space X we consider the *upper Vietoris topology* v, the topology generated by the sets

$$\Box U = \{K \in \mathfrak{P}X \mid K \subseteq U\},$$

where U ranges over the open subsets of X. Since

$$\Box(U \cap V) = \Box U \cap \Box V,$$

the sets $\Box U$ form indeed a basis for the upper Vietoris topology. Equivalently, the sets $\Diamond C = \{K \in \mathfrak{P}X \mid K \cap C \neq \emptyset\}$ are closed for all closed sets C of X and they form a basis for the closed sets of the upper Vietoris topology. The canonical map $\eta_X = (x \mapsto \uparrow x)\colon X \to \mathfrak{P}X$ is a topological embedding. The specialization preorder for the upper Vietoris topology on $\mathfrak{P}X$ agrees with the Smyth preorder $A \sqsubseteq B$, i.e., $\uparrow B \subseteq \uparrow A$. We consider several subspaces of $\mathfrak{P}X$:

$\mathfrak{P}_f X$, the space of all nonempty finite subsets of X,

$\mathfrak{K}X$, the space of all nonempty compact subsets,

$\mathfrak{Q}_f X$, the space of all nonempty finitely generated saturated sets $\uparrow F, F \in \mathfrak{P}_f X$, and

$\mathfrak{Q}X$, the space of all nonempty compact saturated subsets of X.

These spaces are always endowed with the upper Vietoris topology, and the specialization preorder is $\sqsubseteq$ as above. The specialization preorder is a partial order only on $\mathfrak{Q}X$ and $\mathfrak{Q}_f X$.

We also have a semilattice operation on $\mathfrak{P}X$, namely $A \sqcap B = A \cup B$, and $\mathfrak{P}_f X$, $\mathfrak{K}X$, $\mathfrak{Q}_f X$, and $\mathfrak{Q}X$ are subsemilattices thereof. The basic open neighborhoods $\Box U$ are filters, that is, $A \sqcap B \in \Box U$ if and only if $A \in \Box U$ and $B \in \Box U$. This implies that the semilattice operation $\sqcap$ is continuous with respect to the upper Vietoris topology.

2.4 Irreducible Sets

Let us collect some known facts about irreducible sets in a topological space X.

Using that A meets U if and only if $A \not\subseteq X \setminus U$ we see:

Fact 2.3 *For a set A in a topological space X, the following are equivalent:*

(i) For any finite family $(C_i)_{i \in F}$ of closed sets: if $A \subseteq \bigcup_{i \in F} C_i$, then $A \subseteq C_i$ for some i.

(ii) For any finite family $(U_i)_{i \in F}$ of open sets: if A meets all U_i, then A meets $\bigcap_{i \in F} U_i$.

A subset A of a topological space X is said to be *irreducible* if it satisfies the equivalent conditions of 2.3.

Fact 2.4 *For a* closed *set A in a topological space, the following are equivalent:*

(i) A is irreducible.

(ii) For any finite family $(C_i)_{i \in F}$ of closed sets: if $A = \bigcup_{i \in F} C_i$, then $A = C_i$ for some i.

Since an open set meets the closure of A iff it meets A, we have:

Fact 2.5 *A set is irreducible iff its closure is irreducible.*

Fact 2.6 *Let $f : X \to Y$ be a continuous map of topological spaces X and Y. If A is irreducible in X, then its image $f(A)$ is irreducible in Y.*

Proof. If $f(A) \subseteq \bigcup_{i \in F} C_i$, then $A \subseteq f^{-1}(\bigcup_{i \in F} C_i) = \bigcup_{i \in F} f^{-1}C_i$, whence $A \subseteq f^{-1}C_i$ for some i in F, and so $f(A) \subseteq C_i$. □

Fact 2.7 *Every directed set is irreducible. (Here, "directed" refers to the specialization preorder.)*

Proof. Let A be a directed set. We use 2.3 (ii). If A meets $U_1, \ldots, U_n$, then there are points x_i in $A \cap U_i$. Since A is directed, there is an upper bound x of $x_1, \ldots, x_n$ in A. Since open sets are upper sets, x is in $A \cap U_1 \cap \cdots \cap U_n$. □

Fact 2.8 *The irreducible sets of a poset P endowed with the Alexandroff topology are exactly the directed sets of P.*

Proof. Directed sets are irreducible by 2.7. For the opposite direction, let A be an irreducible set and $x_1, \ldots, x_n$ be elements of A. Then A meets the upper (= Alexandroff open) sets $\uparrow x_1, \ldots, \uparrow x_n$. Since A is irreducible, $A \cap \uparrow x_1 \cap \cdots \cap \uparrow x_n \neq \emptyset$ follows. Any member of this intersection is a common upper bound of $x_1, \ldots, x_n$ in A. □

3 Rudin's Lemma and its topological variants

In her original paper [13], which is not easily accessible, M. E. Rudin formulated the following theorem: *If $\mathcal{F}$ is a collection of finite subsets of P which is directed and converges to* 1*, then there is a subset of $\bigcup \mathcal{F}$ which is directed and converges to* 1. Here P is a poset with a maximal element 1; a $\sqsubseteq$-directed family $\mathcal{F}$ is said to converge to 1 if $\bigcap_{F \in \mathcal{F}} \uparrow F = \{1\}$, and a directed set D is said to converge to 1 if

$\bigcap_{d \in D} {\uparrow} d = \{1\}$. M. E. Rudin used transfinite induction for the proof. For the use in domain theory a modified version as in Corollary 3.5 has become prominent.

3.1 A topological variant of Rudin's Lemma

The original Rudin Lemma deals with directed sets. Fact 2.8 suggests to replace directed sets by irreducible sets in a topological setting.

Lemma 3.1 (Topological Rudin Lemma) *Let X be a topological space and $\mathcal{A}$ an irreducible subset of $\mathfrak{K}X$ ($\mathfrak{Q}(X)$, $\mathfrak{Q}_f X$, respectively). Any closed set $C \subseteq X$ that meets all members of $\mathcal{A}$ contains an irreducible closed subset A that still meets all members of $\mathcal{A}$.*

Proof. Let $\mathcal{C}$ be the set of all closed subsets of C that meet all members of $\mathcal{A}$. Then $\mathcal{C}$ is not empty as it contains C, and is closed under filtered intersections by 2.1 since all members of $\mathcal{A}$ are compact. By the order-dual of Zorn's Lemma, $\mathcal{C}$ contains a minimal element A. As a member of $\mathcal{C}$, A is closed and meets all members of $\mathcal{A}$. We show that A is irreducible using 2.4 (ii).

So let $A = \bigcup_{i \in F} C_i$ where $(C_i)_{i \in F}$ is a finite family of closed sets. Every K in $\mathcal{A}$ meets A, and therefore some C_i. Hence $\mathcal{A} \subseteq \bigcup_{i \in F} \Diamond C_i$. Since $\mathcal{A}$ is irreducible in $\mathfrak{K}X$ and the sets $\Diamond C_i$ are closed in $\mathfrak{K}X$ (Section 2.3), $\mathcal{A} \subseteq \Diamond C_k$ for some k in I follows by 2.3 (1). Thus C_k meets all members of $\mathcal{A}$, whence C_k is in $\mathcal{C}$ and is a subset of A. By minimality of A in $\mathcal{C}$, $A = C_k$ follows. □

In the previous Lemma 3.1, one may choose $C = X$ so that for every irreducible subset $\mathcal{A}$ of $\mathfrak{K}X$, $\mathfrak{Q}X$ and $\mathfrak{Q}_f X$, respectively, there is an irreducible closed subset of X that meets all members of $\mathcal{A}$.

By 2.7, directed sets are irreducible. Therefore, 3.1 implies the following corollary:

Corollary 3.2 *Let X be a topological space and $\mathcal{A}$ a $\sqsubseteq$-directed family of nonempty compact subsets of X. Any closed set C that meets all members of $\mathcal{A}$ contains an irreducible closed subset A that still meets all members of $\mathcal{A}$.*

Remark 3.3 M. Erné [3, Proposition 3] had already obtained the following equivalent version of Corollary 3.2:

For every filtered collection $\mathcal{A}$ of nonempty compact saturated subsets of a space X, there is an irreducible (closed) subset A meeting all members of $\mathcal{A}$.

In his paper, Erné emphasizes the fact that this result can be proved without using the full strength of Zorn's lemma (as we did in the proof of 3.1), but only the ultrafilter principle. He also avoids the upper powerspace, but rather embeds the space X into its sobrification X^s. The saturations ${\uparrow}_{X^s} K$ in X^s of the $K \in \mathcal{A}$ form a filtered collection of compact saturated sets which has a nonempty intersection. Picking an element a in this intersection, the set $A = X \cap \mathsf{cl}_{X^s}\{a\}$ is a closed irreducible subset of X meeting all members of $\mathcal{A}$. One can also prove this corollary directly by a slight modification of the proof of 3.1. The price for avoiding the upper powerspace is that 3.2 is less general than 3.1 (but still more general than the original Order Rudin Lemma; see below).

3.2 Rudin's Lemma

We now apply Corollary 3.2 to a space arising from a preorder P with the Alexandroff topology. In such a space, closed = lower, irreducible = directed, and compact = finitary, where those sets K are called *finitary* whose up-sets are finitely generated, that is, $\uparrow K = \uparrow F$ for some finite set F. We obtain:

Lemma 3.4 (Order Rudin Lemma) *Let P be a preorder and $\mathcal{F}$ a $\sqsubseteq$-directed family of finitary upper sets of P. Any lower set L that meets all members of $\mathcal{F}$ has a directed lower subset A that still meets all members of $\mathcal{F}$.*

From this version, it is easy to derive A. Jung's version of Rudin's Lemma [9, Theorem 4.11]:

Corollary 3.5 *If $(F_i)_{i\in I}$ is a $\sqsubseteq$-directed family of nonempty finite sets in a poset P, then there is a directed subset A of $\bigcup_{i\in I} F_i$ that meets all F_i.*

Proof. Let Q be the poset $\bigcup_{i\in I} F_i$ with the order inherited from P. Since all F_i are non-empty, Q itself is a lower set that meets all F_i. By 3.4, it has a directed lower subset A that still meets all F_i. □

In Rudin's Lemma it is essential to restrict to collections $\mathcal{F}$ of finite subsets. Indeed, if we take an infinite set M with the discrete order and consider the filter $\mathcal{F}$ of cofinite subsets, then $\mathcal{F}$ is directed for reverse inclusion, but of course there is no directed subset D satisfying $D \cap F \neq \emptyset$ for all $F \in \mathcal{F}$; indeed, the only directed sets are singleton.

3.3 Another variant of Rudin's Lemma

One may ask the following question: Let $(F_i)_{i\in I}$ be a $\sqsubseteq$-directed family of nonempty finite sets of a poset X. Is there a directed subset D of $\bigcup_i F_i$ which intersects each F_i in exactly one point? A positive answer would be a strengthening of Jung's version 3.5 of Rudin's Lemma, which asserts that there is a directed subset D of $\bigcup_i F_i$ which intersects each F_i in at least one point.

The answer to the question above is negative in general. It is not difficult to come up with a finite counterexample. For treelike directed families, there is a positive answer to our question. For this we use a variant of Rado's Selection Lemma due to R. J. Cowen [2, Theorem 3]:

Let $\mathcal{F}$ be a set of partial functions defined on subsets of a set I with the following properties:

(i) $\mathcal{F}$ is of finite character, that is, f belongs to $\mathcal{F}$ if and only if the restriction of f to any finite subset of its domain belongs to $\mathcal{F}$.

(ii) $\{f(i) \mid f \in \mathcal{F}\}$ is finite for each $i \in I$.

(iii) For each finite $J \subseteq I$, there exists an $f \in \mathcal{F}$ whose domain contains J.

Then $\mathcal{F}$ contains a function defined on all of I.

Lemma 3.6 *Let I be a directed poset which is a tree in the sense that the upper set of each $i \in I$ is linearly ordered. Let $(F_i)_{i \in I}$ be a collection of nonempty finite subsets of a poset P such that $F_i \sqsubseteq F_j$ whenever $i \leq j$. Then one may choose $x_i \in F_i$ for every i such that $x_i \leq x_j$ whenever $i \leq j$.*

Proof. We consider the collection $\mathcal{F}$ of order preserving maps f defined on subsets J of I such that $f(i) \in F_i$ for all $i \in J$. The hypotheses (i), (ii), (iii) of the Cowen Lemma are satisfied: Clearly, this collection $\mathcal{F}$ is of finite character. For every finite subset J of I, we can find an order preserving map x from J to $\bigcup_i F_i$ such that $x_j \in F_j$ for all $j \in J$. For this, we may suppose that J has a greatest element j_0. We begin by choosing any $x_{j_0} \in F_{j_0}$. We now look at the immediate predecessors $j_1, \ldots, j_k$ of j_0 in J and we choose $x_{j_\iota} \in F_{j_\iota}$ such that $x_{j_\iota} \leq x_{j_0}$ which is possible, since $\uparrow F_{j_0} \subseteq \uparrow F_{j_\iota}$ for $\iota = 1, \ldots, k$. For each of the j_ι we repeat the same procedure. After finitely many steps we have exhausted the finite set J. We have used that the directed set I is a tree: descending paths in the finite subset J never meet.

We now can apply Cowen's Selection Lemma cited above and we obtain the desired conclusion. □

Remark 3.7 Notice that a directed set which is a tree has cofinal chains; just take $\uparrow x$ for any member x of the tree. Using König's Lemma, the preceding Lemma 3.6 has been proved by Goubault-Larrecq [6, Lemma 4.12] for the case where I is the set of natural numbers with its usual order.

3.4 The Dcpo Case

The Order Rudin Lemma 3.4 has interesting consequences in a dcpo.

Fact 3.8 *Let D be a dcpo and $\mathcal{F}$ a filtered family of nonempty finitely generated upper sets of D. Any Scott-closed set C that meets all members of $\mathcal{F}$ also meets $\bigcap \mathcal{F}$.*

Proof. Let C be a Scott-closed, hence lower set that meets all members of $\mathcal{F}$. By 3.4, it has a directed subset A that still meets all members of $\mathcal{F}$. The least upper bound x of A exists in the dcpo D and is in C since C is Scott-closed. Since A meets all members of $\mathcal{F}$ and since these members are upper sets, the upper bound x of A is in all of them, i.e., x is in $C \cap \bigcap \mathcal{F}$. □

By contraposition and complementing C, one obtains the following:

Corollary 3.9 *Let D be a dcpo and $\mathcal{F}$ a $\sqsubseteq$-directed family of nonempty finite sets of D. If $\bigcap_{F \in \mathcal{F}} \uparrow F$ is a subset of a Scott-open set U, then already some member of $\mathcal{F}$ is a subset of U.*

Note that these two statements are based on considering two different topologies on the underlying set: 3.4 is the instance of the Topological Rudin Lemma for the Alexandroff topology, whereas the derivation of 3.8 and 3.9 from 3.4 is based on the Scott topology.

Corollary 3.10 *Let D be a dcpo and $\mathcal{F}$ a filtered family of nonempty finitary upper sets of D. Then $\bigcap \mathcal{F}$ is a nonempty compact saturated set.*

Proof. Applying 3.8 in the case $C = X$, we see that $\bigcap \mathcal{F}$ is nonempty. In order to show the compactness of $\bigcap \mathcal{F}$, suppose that $(U_i)_i$ is a family of open sets covering $\bigcap \mathcal{F}$. By the previous corollary, some $K \in \mathcal{F}$ is contained in the open set $\bigcap_i U_i$. By the compactness of K, finitely many of the U_i already cover K, hence they also cover $\bigcap \mathcal{F}$. □

3.5 The Sober Case

The Topological Rudin Lemma itself has analogous consequences in a sober space. Recall that a topological space is *sober*, if every irreducible closed subset A is the closure of a uniquely determined point a. Unlike the dcpo case, all arguments are based on a single topology. Thus, the following is not a generalization of 3.8, but a logically unrelated statement.

Proposition 3.11 *Let X be a sober space and $\mathcal{A}$ an irreducible subset of $\mathfrak{K}X$ ($\mathfrak{Q}X$, $\mathfrak{Q}_f X$, respectively). Then any closed subset C of X that meets all members of $\mathcal{A}$ also meets $\bigcap_{K \in \mathcal{A}} \uparrow K$, and if $\bigcap_{K \in \mathcal{A}} \uparrow K$ is a subset of an open set U, then already some member of $\mathcal{A}$ is a subset of U.*

Proof. Let C be a closed set that meets all members of $\mathcal{A}$. By 3.1, it has an irreducible closed subset A that still meets all members of $\mathcal{A}$. Since X is sober, A is the closure of a unique point x, $A = \mathsf{cl}\{x\} = \downarrow x$. Then $x \in A \subseteq C$, and since A meets all members of $\mathcal{A}$, the greatest element x of A belongs to $\uparrow K$ for all $K \in \mathcal{A}$.

The statement about the open set follows by contraposition and complementing the closed set. □

The following lemma is useful in the proof of the subsequent soberness criterion:

Fact 3.12 *Let $\mathcal{A}$ be a set of compact (supercompact) subsets of a topological space X and K an arbitrary subset of X with the property that K is a subset of an open set U iff some member of $\mathcal{A}$ is a subset of U. Then K is compact (supercompact).*

Proof. Let $(U_i)_{i \in I}$ be a directed (arbitrary) family of open sets such that $K \subseteq \bigcup_{i \in I} U_i$. By hypothesis, there is some Q in $\mathcal{A}$ such that $Q \subseteq \bigcup_{i \in I} U_i$. Since Q is compact (supercompact), $Q \subseteq U_k$ holds for some k in I. By the hypothesis again, $K \subseteq U_k$ follows. □

We now can prove the main result in this section:

Theorem 3.13 *For a topological space X, the following are equivalent:*

(i) *X is sober.*

(ii) *If $\mathcal{A}$ is an irreducible set of $\mathfrak{Q}X$ such that $\bigcap \mathcal{A}$ is a subset of an open set U, then already some member of $\mathcal{A}$ is a subset of U.*

(iii) *$\mathfrak{Q}X$ is sober.*

Proof. The implication (i) ⇒ (ii) holds by 3.11. For (ii) ⇒ (iii), let $\mathcal{A}$ be an irreducible closed set in $\mathfrak{Q}X$. By 3.12, $K = \bigcap \mathcal{A}$ is compact, i.e., an element of $\mathfrak{Q}X$. The property $K \in \Box U$, i.e., $K \subseteq U$, is equivalent to $\mathcal{A} \cap \Box U \neq \emptyset$ by (ii). This equivalence proves $\mathsf{cl}_{\mathfrak{Q}X}\{K\} = \mathcal{A}$.

Finally assume $\mathfrak{Q}X$ is sober and let C be an irreducible closed set of X. Then $\mathcal{A} = \mathsf{cl}\{\uparrow x \mid x \in C\}$ is an irreducible closed set of $\mathfrak{Q}X$ by 2.6 ($(x \mapsto \uparrow x) : X \to \mathfrak{Q}X$ is continuous) and 2.5. Since $\mathfrak{Q}X$ is sober, there is a compact saturated set K such that $\mathcal{A} = \mathsf{cl}\{K\}$. Hence $K \in \Box U$ iff $\{\uparrow x \mid x \in C\}$ meets $\Box U$. Therefore, $\{\uparrow x \mid x \in C\}$ and K satisfy the hypothesis of 3.12, whence K is supercompact. By 2.2, $K = \uparrow a$ holds for some a in X. For all open sets U, C meets U iff $\uparrow x \subseteq U$ for some x in C, iff $K = \uparrow a \subseteq U$, iff a in U. This equivalence implies $C = \mathsf{cl}\{a\}$. □

Remark 3.14 (1) In Statement (ii) one may replace the collection $\mathfrak{Q}X$ of all nonempty compact saturated sets by the collection $\mathfrak{K}X$ of all nonempty compact sets.

(2) Statement (ii) of 3.13 implies the corresponding statement for filtered sets $\mathcal{F}$ of compact upper sets: Whenever $\mathcal{F}$ is a filtered collection of compact saturated sets and U an open set such that $\bigcap \mathcal{F} \subseteq U$, then $Q \subseteq U$ for some $\mathcal{Q} \in \mathcal{F}$. In [4, Definition I-1.24.1] a space has been called *well-filtered*, if this latter property holds.

This “filtered” version of 3.11 can be derived from 3.2, the filtered version of the Topological Rudin Lemma. In his PhD thesis [8, Problem 6, p. 120], the first author asked the question whether the “filtered” version of 3.11 is equivalent to soberness. The answer is “no”! Hui Kou [10] exhibited a counterexample. Thus 3.13 shows that the general “irreducible” version of 3.11 is strictly more powerful than the “filtered” version. In [4, Theorem II-1.21] it is shown nevertheless that “well-filtered” implies “sober” for locally compact spaces.

(3) The implication (i) ⇒ (iii) in the previous theorem had already been proven by A. Schalk [14, Lemma 7.20].

4 Quasicontinuous domains

We present an approach to quasicontinuous dcpos by focussing on the poset $\mathfrak{Q}_f X$ of nonempty finitely generated sets and on the poset $\mathfrak{Q}X$ of nonempty compact saturated sets rather than the Scott-open ones. We present simpler proofs of known results and a characterization of those dcpos that are Smyth powerdomains of quasicontinuous domains.

4.1 The way-below relation on finite subsets

Throughout let X be a dcpo. As before, $\mathfrak{Q}_f X$ denotes the collection of all nonempty finitely generated upper sets ordered by $\sqsubseteq$, that is, reverse inclusion. By $F, G, H, \ldots$ we always denote nonempty finite subsets.

Let us recall the definition of the *way-below* relation on an arbitrary poset P. For $x, y \in P$ one writes

$$x \ll y \iff (\ y \leq \bigvee\nolimits^{\uparrow} D \Rightarrow \exists d \in D.\ x \leq d)$$

that is, $x \ll y$ if, for every directed subset D of P such that $y \leq \bigvee^{\uparrow} D$, there is an element $d \in D$ with $x \leq d$, provided that D has a least upper bound in P.

Let us apply this definition to the poset $\mathfrak{Q}_f X$ ordered by reverse inclusion: $\uparrow G \ll \uparrow H$ iff for every $\sqsubseteq$-directed family $(\uparrow F_i)_i$ such that $\bigcap_i \uparrow F_i$ is a finitely generated upper set contained in $\uparrow H$, there is an i such that $F_i \subseteq \uparrow G$.

We will write $G \ll H$ if $\uparrow G \ll \uparrow H$. The following lemma shows that the way-below relation on the poset $\mathcal{Q}_f P$ agrees with the way-below relation defined for finite subsets of a dcpo in [5] and in [4, Definition III-3,1]:

Lemma 4.1 *For nonempty finite subsets of a dcpo X one has $G \ll H$ if and only if, whenever $h \leq \bigvee^{\uparrow} D$ for some $h \in H$ and directed D, then $d \in \uparrow G$ for some $d \in D$.*

Proof. Suppose first that $G \ll H$ according to our definition. Consider a directed set D such that $\bigvee^{\uparrow} D \in {\uparrow}H$. Then the principal ideals ${\uparrow}d$, $d \in D$, form a filtered family of nonempty finitely generated upper sets with $\bigcap_{d\in D} {\uparrow}d = {\uparrow}(\bigvee^{\uparrow} D) \subseteq {\uparrow}H$. Thus, if ${\uparrow}G \ll {\uparrow}H$, there is a $d \in D$ such that $d \in {\uparrow}G$.

Conversely, suppose that $\bigvee^{\uparrow} D \in {\uparrow}H \Rightarrow \exists d \in D.\ d \in {\uparrow}G$. In order to show that ${\uparrow}G \ll {\uparrow}H$, consider any filtered family of nonempty finitely generated upper sets $({\uparrow}F_i)_i$ whose intersection is a finitely generated upper set contained in ${\uparrow}H$. Suppose that none of the F_i is contained in ${\uparrow}G$. Then the $F'_i = F_i \setminus {\uparrow}G$ are nonempty and they still form a $\sqsubseteq$-directed family. By Jung's version 3.5 of Rudin's Lemma, there is a directed set $D \subseteq \bigcup_i F'_i$ such that $D \cap F'_i \neq \emptyset$ for all i. Then $\bigvee^{\uparrow} D \in {\uparrow}F'_i \subseteq {\uparrow}F_i$ for all i, whence $\bigvee^{\uparrow} D \in \bigcap_i {\uparrow}F_i \subseteq {\uparrow}H$. By our hypothesis, this implies $d \in {\uparrow}G$ for some $d \in D$, which contradicts the fact that d belongs to some F'_i which is disjoint from ${\uparrow}G$ by its definition. Thus, some F_i is contained in ${\uparrow}G$. $\square$

We abbreviate $G \ll \{y\}$ by $G \ll y$. As a special case of the previous lemma we obtain:

Corollary 4.2 *$G \ll y$ iff $(y \leq \bigvee^{\uparrow} D \Rightarrow \exists d \in D.\ d \in {\uparrow}G)$.*

In particular, $\{x\} \ll \{y\}$ in $\mathfrak{Q}_f X$ iff $x \ll y$ in X. Thus the canonical map $x \mapsto {\uparrow}x \colon X \to \mathfrak{Q}_f X$ is an embedding for the order, for directed suprema and for $\ll$. Also note that $G \ll H$ iff $G \ll y$ for all $y \in H$.

4.2 Quasi-continuous dcpos

Recall that a poset P is called continuous if, for all $x \in P$, the set of all $y \ll x$ is directed and $x = \bigvee^{\uparrow}\{y \mid y \ll x\}$. We now define:

Definition 4.3 A dcpo X is called *quasicontinuous* if the poset $\mathfrak{Q}_f X$ of nonempty finitely generated upper sets ordered by reverse inclusion $\sqsubseteq$ is continuous.

In the following proposition we show that our definition of quasicontinuity is equivalent to the one given in [5] and [4, Definition III-3.2]:

Proposition 4.4 *A dcpo X is quasicontinuous iff (*) for every $x \in X$ the family of nonempty finite sets $F \ll x$ is $\sqsubseteq$-directed and $\bigcap_{F \ll x} {\uparrow}F = {\uparrow}x$, that is, for all $y \not\geq x$ there is a finite $F \ll x$ such that $y \notin {\uparrow}F$.*

Proof. Suppose first that X is quasicontinuous according to our definition, that is, $(\mathfrak{Q}_f X, \sqsubseteq)$ is a continuous poset. Then the $F \ll x$ form a $\sqsubseteq$-directed subset of $\mathfrak{Q}_f(P)$ and ${\uparrow}x = \bigcap\{{\uparrow}F \mid F \ll x\}$.

Suppose conversely that condition (*) is satisfied. As we have remarked, we have $F \ll G$ iff $F \ll x$ for all $x \in G$. By hypothesis, the set of $F \ll x$ is a $\sqsubseteq$-ideal.

In a semilattice, an intersection of finitely many ideals is an ideal. Thus, the set of $F \ll G$ is $\sqsubseteq$-directed. Further $\bigcap_{F \ll G} \uparrow F = \bigcup_{x \in G} \bigcap_{F \ll x} \uparrow F = \bigcup_{x \in G} \uparrow x$ by condition (*), $= \uparrow G$. □

We deduce some properties of quasicontinuous dcpos X:

Properties 4.5 *Let X be a quasicontinuous dcpo.*

(i) *The way-below relation $F \ll G$ on $\mathfrak{Q}_f X$ has the interpolation property. In particular, if $F \ll x$, then there is a G such that $F \ll G \ll x$.* (Compare [4, Proposition III-3.5].)

Indeed, by definition $\mathfrak{Q}_f X$ is a continuous poset, and the way-below relation on every continuous poset has the interpolation property.

(ii) *A subset U of X is Scott-open if and only if, for every $x \in U$, there is a nonempty finite set $F \ll x$ such that $\uparrow F \subseteq U$.* (Compare [4, Proposition III-3.6].)

Proof. Let U be a Scott-open subset of X and $x \in U$. We know that $\uparrow x = \bigcap_{F \ll x} \uparrow F$. Since the collection of $F \ll x$ is $\sqsubseteq$-directed, Corollary 3.9 tells us that there is an $F \ll x$ such that $F \subseteq U$. Suppose conversely that for every $x \in U$ there is a finite set $F \ll x$ such that $\uparrow F \subseteq U$. In order to show that U is Scott-open, notice first that U is an upper set; indeed, for every $x \in U$ there is a finite set $F \ll x$ such that $\uparrow F \subseteq U$, whence $\uparrow x \subseteq U$. Now, consider any directed family $(x_i)_i$ such that $\bigvee^{\uparrow} x_i \in U$. By hypothesis there is a finite $F \ll \bigvee^{\uparrow}_i x_i$ such that $\uparrow F \subseteq U$; hence there is an element $x \in F$ and an i such that $x \leq x_i$, whence $x_i \in U$. □

(iii) *For every nonempty finite subset F, the set $\{x \in X \mid F \ll x\}$ is the interior of $\uparrow F$ for the Scott topology and, hence, Scott open. Moreover, the sets of the form $\{x \in X \mid F \ll x\}$, F finite, form a basis for the Scott topology on X.* (Compare [4, Proposition III-3.6].)

Proof. For a nonempty finite set F let $U = \{x \in X \mid F \ll x\}$. By the interpolation property, for every $x \in U$ there is an F' such that $F \ll F' \ll x$. Then $F \ll x'$ for every $x' \in F'$, whence $F' \subseteq U$. By the previous item we conclude that U is Scott-open. Further, U is the interior of $\uparrow F$. Let indeed x be an element in the interior of $\uparrow F$. There is a finite set F' in $\mathsf{int}\uparrow F$ such that $F' \ll x$. Then also $F \ll x$, whence $x \in U$. □

(iv) *Every nonempty compact saturated subset Q of X has a neighborhood basis of finitely generated upper sets.*

Proof. Let Q be nonempty, compact and saturated. Let U be a Scott-open set containing Q. By property (ii), for $x \in Q$ we may choose a finite set $F_x \subseteq U$ such that $F_x \ll x$. By property (iii), $\uparrow F_x$ is a neighborhood of x for the Scott topology. Since Q is compact, finitely many of those neighborhoods cover Q. Thus there is a finite subset F in U such that $\uparrow F$ is a neighborhood of Q. Thus, Q has a neighborhood basis of finitely generated upper sets. □

(v) *A quasicontinuous dcpo X is locally compact for its Scott topology.* (Compare [4, Proposition III-3.7(a)].)

By (iv) every $x \in X$ has a neighborhood basis of finitely generated upper sets and those are compact.

(vi) *On $\mathfrak{Q}X$, the upper Vietoris topology agrees with the Scott topology.* (Compare [14, Lemma 7.26][6, Corollary 3.6].)

Proof. The basic open sets for the upper Vietoris topology, $\Box U$ for Scott-open $U \subseteq X$, are also Scott-open in $\mathfrak{Q}X$. Indeed if $(\uparrow F_i)$ is a $\sqsubseteq$-directed family such that $\bigcap_i \uparrow F_i \subseteq U$, then $\uparrow F_i \subseteq U$ for some i by Corollary 3.9.

Conversely, a basic open set of the Scott topology on $\mathfrak{Q}X$ is of the form $\{Q \in \mathfrak{Q}X \mid \uparrow F \ll Q\}$ and this set can be rewritten as $\Box V$ where $V = \{x \in X \mid F \ll x\}$ is Scott-open by (iii). □

(vii) *A quasicontinuous dcpo X is sober.* (Compare [4, Proposition III-3.7].)

Indeed, $\mathfrak{Q}X$ is a continuous dcpo, hence sober for its Scott topology. Since the Scott topology agrees with the upper Vietoris topology (vi), X is sober by Theorem 3.13.

Let us draw some conclusions for the powerdomain $\mathfrak{Q}X$ of all nonempty compact saturated subsets:

Proposition 4.6 *Let X be a quasicontinuous dcpo. Then*

(1) *$\mathfrak{Q}X$ is a continuous semilattice with respect to the operation $K \sqcap K' = K \cup K'$,*

(2) *the nonempty finitely generated upper sets form a basis $\mathfrak{Q}_f X$ of $\mathfrak{Q}X$,*

(3) *the semilattice operation preserves the way-below relation in the sense that $K \ll Q$, $K' \ll Q'$ imply $K \sqcap K' \ll Q \sqcap Q'$.*

Moreover, the canonical embedding $\eta_X = (x \mapsto \uparrow x)\colon X \to \mathfrak{Q}X$ is an embedding for the respective Scott, lower and Lawson topologies.

Proof. Since X is locally compact for the Scott topology by 4.5(v), $\mathfrak{Q}X$ is a continuous dcpo in which $K \ll Q$ iff K is a neighborhood of Q by [14, Proposition 7.25]. The finitely generated upper sets form a basis by 4.5(iv). Property (3) is a consequence of the fact that if K is a neighborhood of Q and K' a neighborhood of Q', then $K \cup K'$ is a neighborhood of $Q \cup Q'$.

The map $\eta_X = (x \mapsto \uparrow x)\colon X \to \mathfrak{Q}X$ is an embedding of X (with the Scott topology) into $\mathfrak{Q}X$ with the upper Vietoris topology which agrees with the Scott topology by 4.5(vi).

The map η_X is also an embedding for the respective lower topologies: Since every compact saturated set is the intersection of a filtered family of finitely generated upper sets, a subbasis for the closed sets of the lower topology on $\mathfrak{Q}X$ is given by the sets of the form $\{Q \in \mathfrak{Q}X \mid Q \subseteq \uparrow F\}$, where F ranges over the finite subsets of X. The inverse image of such a set under η_X is the set $\{x \in X \mid \uparrow x \subseteq \uparrow F\} = \uparrow F$, and these sets form a basis for the closed sets for the lower topology on X. □

Since the Lawson topology on the continuous dcpo $\mathfrak{Q}X$ is regular and Hausdorff, these properties are inherited by the Lawson topology on X. (Compare [4, Proposition III-3.7(b)].)

4.3 Abstract characterization of the domains $\mathfrak{Q}X$ for quasicontinuous X

We intend to show that the properties (1), (2), and (3) in Proposition 4.6 characterize those dcpos that are (isomorphic to) the powerdomain of all compact saturated subsets of quasicontinuous dcpos.

For this we have to identify X in $\mathfrak{Q}X$. In $\mathfrak{Q}X$ we can find the elements x of X through the sets of the form $\uparrow x$. Can we distinguish these particular compact saturated sets from the others in the domain $\mathfrak{Q}X$ by an intrinsic property?

Recall that an element p of a meet-semilattice is called *prime* if $x \wedge y \le p$ implies $x \le p$ or $y \le p$. If there is a top element, we consider it to be prime as in [4]. The property of being prime extends from finite meets to meets of compact sets:

Lemma 4.7 *If p is a prime element in a quasi-continuous meet-semilattice S and Q a Scott-compact subset of S with a greatest lower bound $\bigwedge Q$ in S then $\bigwedge Q \le p$ implies that $q \le p$ for some $q \in Q$.*

Proof. Assume $q \not\le p$ for all $q \in Q$. Then for all q in Q, there is a finite $F_q \ll q$ such that $p \notin \uparrow F_q$. The sets $\{x \mid F_q \ll x\}$, $q \in Q$, form an open cover of Q. By compactness, there is a finite $G \subseteq Q$ such that $Q \subseteq \bigcup_{q \in G}\{x \mid F_q \ll x\}$. Let F be the finite set $\bigcup_{q \in G} F_q$. Then $Q \subseteq \uparrow F$, and so $p \ge \bigwedge Q \ge \bigwedge F$. Since p is prime, there is some a in F such that $p \ge a$, whence there is some q in G such that $p \in \uparrow F_q$ – a contradiction. □

We use this lemma for the following:

Lemma 4.8 *Let X be a quasicontinuous dcpo. The prime elements of the $\sqcap$-semilattice $\mathfrak{Q}X$ are the principal filters $\uparrow x, x \in X$.*

Proof. All the $\uparrow x$, $x \in X$, are prime in $\mathfrak{Q}X$. Indeed $\uparrow x \subseteq K_1 \cup K_2$ implies $x \in K_1$ or $x \in K_2$, whence $\uparrow x \subseteq K_1$ or $\uparrow x \subseteq K_2$. It remains to show that every prime element in $\mathfrak{Q}X$ is of the form $\uparrow x$ for some $x \in X$.

Consider $K \in \mathfrak{Q}X$. The set $\mathcal{K} = \{\uparrow x \mid x \in K\}$ is a compact subset of $\mathfrak{Q}X$. Its union is K, so $\mathcal{K}$ has an infimum $K = \bigsqcap \mathcal{K}$ in $\mathfrak{Q}X$. We now use Lemma 4.7: If K is prime in $\mathfrak{Q}X$, then there is an element $\uparrow x \in \mathcal{K}$ such that $\uparrow x \sqsubseteq K$, which implies that $K = \uparrow x$ for some $x \in K$. □

We now can formulate our representation theorem:

Theorem 4.9 *Suppose that*

(1) *L is a continuous directed complete $\wedge$-semilattice,*

(2) *the finite meets of prime elements form a basis of L,*

(3) *the way-below relation $\ll$ on L is preserved by the semilattice operation $\wedge$, that is, if $a \ll b$ and $a' \ll b'$ then $a \wedge a' \ll b \wedge b'$.*

Then the prime elements of L form a quasicontinuous dcpo X in the induced order and L is isomorphic to the continuous $\sqcap$-semilattice of all compact saturated subsets of X.

We will prove the theorem in several steps. For this we use two relaxed notions of primeness. An ideal I of a $\wedge$-semilattice is called prime if $a \wedge b \in I$ implies $a \in I$ or $b \in I$. An element p is called *pseudoprime* if there is a prime ideal I such that $p = \bigvee^{\uparrow} I$. Further, p is called *weakly prime* if $x_1 \wedge \ldots \wedge x_n \ll p$ implies $x_i \leq p$ for some i (compare [4, Definition V-3.1 and Lemma V-3.4]).

Clearly prime elements are pseudoprime and weakly prime. By [4, Proposition I-3.28] we have:

Lemma 4.10 *Let L be a continuous directed complete $\wedge$-semilattice. Suppose that $\wedge$ preserves the way-below relation in L. Then the pseudoprime elements agree with the prime elements.*

In order to prove that weakly prime elements are pseudoprime in our setting we use distributivity. The following definition of distributivity for semilattices agrees with the usual definition of distributivity when applied to lattices:

Definition 4.11 A $\wedge$-semilattice is *distributive* if the following property holds for all a, b, and x: If $a \wedge b \leq x$, then there exist $a' \geq a$ and $b' \geq b$ such that $a' \wedge b' = x$.

By [4, Corollary I-3.13], in a distributive continuous $\wedge$-semilattice every element is a meet of primes. Let us remark:

Lemma 4.12 *If the directed complete $\wedge$-semilattice L satisfies the three hypotheses of Theorem 4.9, then L is a distributive semilattice.*

Proof. We first consider the basis L_f of L consisting of all finite meets of prime elements. Clearly, L_f is a subsemilattice of L. We show that L_f is distributive. Let a, b, x be elements of L_f such that $a \wedge b \leq x$. Let G, H, F be finite sets of prime elements such that $a = \bigwedge G$, $b = \bigwedge H$, $x = \bigwedge F$. We first claim that $F \subseteq \uparrow(G \cup H)$. Indeed, as $a \wedge b \leq x$, we have $\bigwedge(G \cup H) = \bigwedge G \wedge \bigwedge H \leq \bigwedge F \leq p$ for every $p \in F$; since p is prime, there is a $q \in G \cup H$ such that $q \leq p$. Now let $G' = F \cap \uparrow G$ and $H' = F \cap \uparrow H$ and $a' = \bigwedge G', b' = \bigwedge H'$. Then $a' \wedge b' = \bigwedge(F \cap \uparrow G) \wedge \bigwedge(F \cap \uparrow H) = \bigwedge((F \cap \uparrow G) \cup (F \cap \uparrow H)) = \bigwedge(F \cap (\uparrow G \cup \uparrow H)) = \bigwedge F = x$.

It is more or less routine to check now that L itself is a distributive semilattice.□

We will need the following separation property:

Lemma 4.13 *Let F be a filter and J an ideal disjoint from F in a distributive $\wedge$-semilattice L. Then there is a prime ideal I containing J but still disjoint from F.*

This is Lemma 2 in [7, Section II.5]. For a proof [7] only refers to the corresponding proof for distributive lattices, although in the case of semilattices the proof is slightly more sophisticated. An explicit proof can be found, for example, in [1].

Lemma 4.14 (compare [4, Proposition I-3.25]) *In a continuous directed complete distributive $\wedge$-semilattice L, the pseudoprime elements agree with the weakly prime elements.*

Proof. Let p be pseudoprime and I a prime ideal with $p = \bigvee^{\uparrow} I$. Suppose $x_1 \wedge \ldots \wedge x_n \ll p$. Then $x_1 \wedge \ldots \wedge x_n \in I$ which implies $x_i \in I$ for some i. Thus, $x_i \leq \bigvee^{\uparrow} I = p$ which shows that p is weakly prime.

Suppose now that L is distributive and p weakly prime. Let F be the filter generated by $L \setminus \downarrow p$. Since p is weakly prime, F is disjoint from the ideal $\twoheaddownarrow p = \{a \in L \mid a \ll p\}$. By distributivity and Lemma 4.13, there is a prime ideal I containing the ideal $\twoheaddownarrow p$ disjoint from F, hence contained in $\downarrow p$. Thus $p = \bigvee^{\uparrow} \twoheaddownarrow p \leq \bigvee^{\uparrow} I \leq p$, whence $p = \bigvee^{\uparrow} I$, that is, p is pseudoprime. □

Proof of Theorem 4.9. Suppose that L satisfies the hypotheses of the theorem. Let X be the set of prime elements of L. Under our hypotheses the notions prime, weakly prime and pseudoprime agree by Lemma 4.10 and Lemma 4.14. We conclude that the join of a directed set D of prime elements is prime; indeed, $\downarrow D$ is a prime ideal, whence $\bigvee^{\uparrow} D$ is pseudoprime and consequently prime.

Now let L_f be the set of all finite meets of primes in L. We firstly notice that, for every $x \in L$, the sets $\twoheaduparrow f = \{f \in L_f \mid f \ll x\}$ form a neighborhood basis with respect to the Scott topology. Indeed, if U is a Scott open neighborhood of $x \in L$, there is an $x' \in U$ with $x' \ll x$. Since x' is the sup of a directed family of elements in L_f, there is an element $f \in L_f$ with $f \in U$ and $f \le x'$ whence $\twoheaduparrow f$ is a Scott-open neighborhood of x contained in U.

Now look at a $p \in X$ and a finite subset F of X such that $\bigwedge F \ll p$ in L. We conclude that $F \ll \{p\}$ in X. Indeed, if D is a directed set in X such that $p \le \bigvee^{\uparrow} D$, then there is a $d \in D$ such that $\bigwedge F \le d$ which implies that $x \le d$ for some $x \in F$, since d is prime. The set of all $f \in L_f$ such that $f \ll p$ is directed, since the $\twoheaduparrow f, f \ll p, f \in L_f$ form a neighborhood base of p in L. Let q be a prime element with $p \not\le q$. There is an $f = \bigwedge F \in L_f$ such that $f \ll p$ but $f \not\le q$. Thus $F \ll \{p\}$ in X but $q \notin {\uparrow_X} F$. This shows that X is a quasicontinuous dcpo.

We now have to show that L is isomorphic to the domain $\mathfrak{Q}X$ of Scott-compact saturated subsets of X. For every $a \in L$ consider the saturated subset ${\uparrow} a \cap X$ of X. Suppose first $a \in L_f$. Then $a = p_1 \wedge \ldots \wedge p_n$ for prime elements $p_1, \ldots, p_n \in X$. For any $p \in X$, one has $p \ge a$ iff $p \ge p_i$ for some i. Thus, ${\uparrow} a \cap X$ is the upper set in X generated by the finite set $\{p_1, \ldots, p_n\}$, hence a compact saturated subset of X. An arbitrary $a \in L$ is the sup of the directed family of elements f_j in L_f with $f_j \ll a$. Then ${\uparrow} a \cap X$ is the intersection of the filtered family ${\uparrow} F_j \cap X$ of finitely generated upper sets in X, hence compact and saturated by 3.10. Thus $a \mapsto {\uparrow} a \cap X$ is a map from L into $\mathfrak{Q}X$, which clearly is order preserving.

Conversely, let K be a Scott-compact saturated subset of X. Then K is the intersection of the filtered family ${\uparrow} F_j$ of finitely generated upper sets in X such that $F_j \ll K$. We assign to K the element $\bigvee^{\uparrow}_j \bigwedge F_j$ of L and we have a map from $\mathfrak{Q}X$ to L which also is clearly order preserving.

It is straightforward to check that these two maps are inverse to each other, and the proof is complete. □

References

[1] Celani, S. A., *Topological representation of distributive semilattices*, Scientiae Mathematicae Japonicae Online **8** (2003), 41–51.

[2] Cowen, R. J., *Some combinatorial theorems equivalent to the prime ideal theorem*, Proceedings of the American Mathematical Society **41** (1973), 268–273.

[3] Erné, M., *Sober spaces, well-filtration and compactness principles.* Preprint, Universität Hannover, http://www.iazd.uni-hannover.de/~erne/preprints/sober.pdf

[4] Gierz, G., K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove, and D. S. Scott, "Continuous Lattices and Domains," Encyclopedia of Mathematics and its Applications, vol.**93**, Cambridge University Press, 2003, xxxvi+591 pages.

[5] Gierz, G., J. D. Lawson, and A. R. Stralka, *Quasicontinuous posets,* Houston Journal of Mathematics **9**(2) (1983), 191–208.

[6] Goubault-Larrecq, J., *QRB-domains and the probabilistic powerdomain,* Logical Methods in Computer Science **8**(1:14) (2012), 1–33.

[7] Grätzer, G., “Lattice Theory: Foundations,” Birkhäuser, 2011.

[8] Heckmann, R., “Power Domain Constructions,” PhD Dissertation, Universität des Saarlandes, 1990. http://rw4.cs.uni-sb.de/~heckmann/diss/diss.html

[9] Jung, A., “Cartesian Closed Categories of Domains,” CWI Tracts vol. **66**, Centrum voor Wiskunde en Informatica, Amsterdam 1989, 107 pp.

[10] Kou, Hui, *Uk-admitting dcpo's need not be sober.* In K. Keimel, G.-Q. Zhang, Y.-M. Liu, and Y.-X. Chen (eds.), “Domains and Processes,” Proc. of the First Int. Conf. on Domain Theory, Shanghai 1999, Semantic Structures in Computation vol. **1**, pages 41-50. Kluwer, 2001.

[11] Li G. and L. Xu, *QFS-Domains and their Lawson compactness.* Order **30** (2013), 233–248.

[12] Lystad, G. S., “Compact Zero-dimensional Semilattices,” Ph.D. Dissertation, University of California, Riverside, 1978.

[13] Rudin, M. E., *Directed sets which converge,* Proc. Riverside Symposium on Topology and Modern Analysis, 1980, 305–307.

[14] Schalk, A., “Algebras for Generalized Power Constructions,” PhD thesis, Technische Universität Darmstadt, 1993. http://www.cs.man.ac.uk/~schalk/publ/index.html

[15] Smyth, M. B., *Powerdomains,* Journal of Computer and System Sciences **16** (1978), 23–36.

[16] Smyth, M. B., *Powerdomains and predicate transformers: a topological view,* In J. Diaz (ed.), “Automata, Languages and Programming,” Lecture Notes in Computer Science **154**, 662–675. Springer Verlag, 1983.

[17] Venugopalan, P., *Quasicontinuous posets,* Semigroup Forum **41** (1990), 193–200.

On Block Structures in Quantum Computation

Bart Jacobs

Institute for Computing and Information Sciences (iCIS),
Radboud University Nijmegen, The Netherlands.
Web address: www.cs.ru.nl/B.Jacobs

June 24, 2013

Abstract

A block is a language construct in programming that temporarily enlarges the state space. It is typically opened by initialising some local variables, and closed via a return statement. The “scope” of these local variables is then restricted to the block in which they occur. In quantum computation such temporary extensions of the state space also play an important role. This paper axiomatises “logical” blocks in a categorical manner. Opening a block may happen via a measurement, so that the block captures the various possibilities that result from the measurement. Following work of Coecke and Pavlović we show that von Neumann projective measurements can be described as an Eilenberg-Moore coalgebra of a comonad associated with a particular kind of block structure. Closing of a block involves a collapse of options. Such blocks are investigated in non-deterministic, probabilistic, and quantum computation. In the latter setting it is shown that there are two block structures in the category of C^*-algebras, via copowers and via matrices.

Keywords: Block structure, non-deterministic, probabilistic, quantum program semantics, effect logic

1 Introduction

In imperative programming languages one may find block structures of the form:

$$\{\texttt{int v = 0; ...; return}\} \tag{1}$$

Such a block is a temporary extension of the state space. It is “opened” by initialisation of some variables, and “closed” by a return statement. Although quantum programming is still in an embryonic state, it is clear, at least at the abstract level, that some sort of block structure is essential. For instance, in [9, Corollary 4.19] one finds that each completely positive map $S\colon \mathcal{DM}(H) \to \mathcal{DM}(H)$ between density matrices on a Hilbert space H — the interpretation of a quantum program — is of the form:

$$S(\rho) = tr_K\big(U(\rho \otimes \xi)U^{\dagger}\big),$$

[1] bart@cs.ru.nl

where U is unitary operator on a state space $H \otimes K$ enlarging H with an "ancilla" space K, ξ is a pure state $|\,v\,\rangle\langle\,v\,|$ for some vector $|\,v\,\rangle \in K$, and tr_K is the partial trace operation. Essentially, this normal form result is based on Stinespring's Theorem (see *loc. cit.*). Here we see, similar to (1), extension of the state with K, opening of this block via the initial value ξ, and closing of the block via the partial trace tr_K.

In this paper we explore block structures at a more elementary level. They are defined as a collection of endofunctors $\mathcal{B}_n \colon \mathbf{A} \to \mathbf{A}$, for natural numbers $n > 0$, on a category $\mathbf{A}$, together with "in" and "out" maps for opening and closing a block. A "logical" block structures comes equipped with "characteristic" or "measurement" maps $X \to \mathcal{B}_n(X)$, induced by n-tests of predicates. Such maps can also open a block structure, via the various options that result from measurement. These logical block structures will be described in various categories, for non-deterministic, probabilistic (both discrete and continuous), and quantum computation. Interestingly, on Hilbert spaces with their standard logic of effects, there is no logical block structure, because there is no operation for closing blocks. This structure does exist on C^*-algebras. Hence, not directly on a Hilbert space H, but on the associated C^*-algebra $\mathcal{L}(H)$ of endomaps, we find the relevant logical block structure.

In the final section we use these logical block structures to give a diagrammatic description of two familiar quantum protocols, namely superdense coding and teleportation. The ultimate goal is to develop an appropriate logic for such protocols and to formalise the representation in a computer algebra tool, for simulation and verification. Thus, the paper follows earlier work on semantics of quantum programming languages, like, for instance [1,19,20,21] and [7].

Among the logical predicates that we use there is a subclass of "projections", with as typical property that iterated measurements give the same outcome. In [3] it was noticed that this property (of von Neumann projective measurements) is captured categorically by the "δ-law" for an Eilenberg-Moore coalgebra c, which requires $\delta \circ c = T(c) \circ c$, where T is the comonad involved; this corresponds to the requirement $P_i P_j = \delta_{ij} P_i$. The other equality that such a coalgebra must satisfy, namely $\varepsilon \circ c = \mathrm{id}$, corresponds to the condition $\sum_i P_i = 1$. The block structures that we use here allow us to generalise this approach in several directions, by showing that it also:

- occurs in simpler situations than quantum models, namely in non-deterministic and in probabilistic models, represented by the Kleisli categories of the powerset monad $\mathcal{P}$, and of the distribution and Giry monads $\mathcal{D}$ and $\mathcal{G}$;
- extends to C^*-algebras, but only in the commutative case, for one of the available block structures, namely the "copower" one that forms a comonad.

This paper unveils two block structures on C^*-algebras: one given by copowers and one by matrices. At this stage it fails to provide an answer to the question whether one of them is the right one, and in which sense? This will require more research.

2 Block structures

This section contains the basic definition of a block structure as a collection of endofunctors indexed by natural numbers, and also some examples. It starts with a very basic result describing some of the relevant endofunctors as (co)monads.

We shall write + for a coproduct in a category, with coprojections $\kappa_i \colon X_i \to X_1 + X_2$ and cotupling $[f_1, f_2] \colon X_1 + X_2 \to Y$, for $f_i \colon X_i \to Y$. For maps $g_i \colon X_i \to Y_i$ there is the coproduct of maps $g_1 + g_2 = [\kappa_1 \circ g_1, \kappa_2 \circ g_2] \colon X_1 + X_2 \to Y_1 + Y_2$. Dually, we write products as $\times$, with projections $\pi_i \colon X_1 \times X_2 \to X_i$ and tupling $\langle f, g\rangle \colon Y \to X_1 \times X_2$.

Lemma 2.1 *Let $\boldsymbol{C}$ be a category with coproducts +. For each natural number $n > 0$, the n-fold copower functor $n \cdot (-) \colon \boldsymbol{C} \to \boldsymbol{C}$ is a comonad, where*

$$n \cdot X = \underbrace{X + \cdots + X}_{n\ \textit{times}}$$

The counit $\varepsilon \colon n \cdot X \to X$ and comultiplication $\delta \colon n \cdot X \to n \cdot (n \cdot X)$ are given by:

$$\varepsilon = \nabla = [\mathrm{id}, \ldots, \mathrm{id}] \qquad \delta = \kappa_1 + \cdots + \kappa_n = [\kappa_i \circ \kappa_i]_{i \leq n}.$$

Dually, in presence of products $\times$, the n-fold power functor $(-)^n$ is a monad on $\boldsymbol{C}$, with unit $\eta = \Delta = \langle \mathrm{id}, \ldots, \mathrm{id}\rangle \colon X \to X^n$ and multiplication $\mu = \langle \pi_i \circ \pi_i\rangle_{i \in n} \colon (X^n)^n \to X^n$. □

On an abstract level these (co)monad structures arise because the n-element set n carries a comonoid structure $1 \xleftarrow{!} n \xrightarrow{\langle \mathrm{id}, \mathrm{id}\rangle} n \times n$. But on a more concrete level, it is not hard to verify the comonad equations $\varepsilon \circ \delta = \mathrm{id} = (n \cdot \varepsilon) \circ \delta$ and $\delta \circ \delta = (n \cdot \delta) \circ \delta$.

Definition 2.2 A *block structure* on a category $\mathbf{A}$ consists of a collection of endofunctors $\mathcal{B}_n \colon \mathbf{A} \to \mathbf{A}$, for $n > 0$, with natural isomorphisms

$$\mathcal{B}_1(X) \cong X \qquad \text{and} \qquad \mathcal{B}_m(\mathcal{B}_n(X)) \cong \mathcal{B}_{m \times n}(X), \tag{2}$$

and with two collections of natural transformations $in_n \colon \mathrm{Id} \Rightarrow \mathcal{B}_n$ and $out_n \colon \mathcal{B}_n \Rightarrow \mathrm{Id}$ with $out_n \circ in_n = \mathrm{id}$, as in:

$$\begin{tikzcd}
X \arrow[r, "in_n"] \arrow[dr, equal] & \mathcal{B}_n(X) \arrow[d, "out_n"] \\
& X
\end{tikzcd}$$

For the comonad $X \mapsto n \cdot X$ and monad $X \mapsto X^n$ from Lemma 2.1 there are obvious isomorphisms as in (2), namely:

$$1 \cdot X \cong X \qquad m \cdot (n \cdot X) \cong (m \times n) \cdot X \qquad X^1 \cong X \qquad (X^n)^m \cong X^{m \times n}.$$

One can turn the copower $n \cdot (-)$ into a block structure by choosing the first coprojection $\kappa_1 \colon X \to n \cdot X$ as "in". This however, looks rather arbitrary. In the next example we see that a more natural option exists in a quantitative setting, as given by the Kleisli category of the distribution monad $\mathcal{D}$. We recall that $\mathcal{D}$ is the (finite discrete) distribution monad $\mathcal{D}\colon \mathbf{Sets} \to \mathbf{Sets}$, given by formal finite convex sums:

$$\mathcal{D}(X) = \{\varphi\colon X \to [0,1] \mid \mathit{supp}(\varphi) \text{ is finite, and } \textstyle\sum_x \varphi(x) = 1\}.$$

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 $\eta^{\mathcal{D}}(x) = 1x$ and by matrix multiplication: $\mu^{\mathcal{D}}(\Phi)(x) = \sum_\varphi \Phi(\varphi)\cdot\varphi(x)$.

Example 2.3 The Kleisli category $\mathcal{K}\ell(\mathcal{D})$ of the distribution monad $\mathcal{D}\colon \mathbf{Sets} \to \mathbf{Sets}$ inherits coproducts $+$ from $\mathbf{Sets}$, so that $X \mapsto n \cdot X$ is a comonad, following Lemma 2.1. We can turn $n \cdot (-)$ into a block structure via an "in" map in $\mathcal{K}\ell(\mathcal{D})$, namely

$$X \xrightarrow{\mathit{in}_n} n \cdot X \qquad \text{given by} \qquad x \longmapsto \tfrac{1}{n}\kappa_1 x + \cdots + \tfrac{1}{n}\kappa_n x.$$

Thus, $\mathit{in}_n(x) \in \mathcal{D}(X)$ defines a uniform distribution over the various coprojections $\kappa_i x \in n \cdot X$. Taking the counit $\varepsilon = \nabla\colon n \cdot X \to X$ as "out" map we get a block structure. Writing Kleisli composition as $g \odot f = \mu^{\mathcal{D}} \circ \mathcal{D}(g) \circ f$, we have:

$$\begin{aligned}
\big(\nabla \odot \mathit{in}_n\big)(x) &= \big(\mu^{\mathcal{D}} \circ \mathcal{D}([\eta^{\mathcal{D}}, \ldots, \eta^{\mathcal{D}}])\big)\big(\textstyle\sum_i \frac{1}{n}\kappa_i x\big) \\
&= \mu^{\mathcal{D}}\big(\textstyle\sum_i \frac{1}{n}([\eta^{\mathcal{D}}, \ldots, \eta^{\mathcal{D}}] \circ \kappa_i)x\big) \\
&= \mu^{\mathcal{D}}\big(\textstyle\sum_i \frac{1}{n}1x\big) \;=\; \mu^{\mathcal{D}}\big(1(1x)\big) \;=\; 1x \;=\; \eta^{\mathcal{D}}(x) \;=\; \mathrm{id}(x).
\end{aligned}$$

The product case $X \mapsto X^n$ is a bit more subtle, because $\times$ is a tensor, not a cartesian product, on $\mathcal{K}\ell(\mathcal{D})$. But since $\mathcal{D}(1) = 1$, the tensor unit is the terminal object 1, so we have a tensor with projections. This allows us to define η and μ as in Lemma 2.1. An associated "out" map can be defined, again via a uniform distribution:

$$X^n \xrightarrow{\mathit{out}_n} X \qquad \text{namely} \qquad (x_1, \ldots, x_n) \longmapsto \tfrac{1}{n}x_1 + \cdots + \tfrac{1}{n}x_n$$

Then:

$$\begin{aligned}
\big(\mathit{out}_n \odot \eta\big)(x) &= \big(\mu^{\mathcal{D}} \circ \mathcal{D}(\mathit{out}_n)\big)\big(1(x, \ldots, x)\big) \\
&= \mu^{\mathcal{D}}\big(1\mathit{out}_n(x, \ldots, x)\big) \\
&= \mu^{\mathcal{D}}\big(1(\tfrac{1}{n}x + \cdots + \tfrac{1}{n}x)\big) \;=\; \mu^{\mathcal{D}}\big(1(1x)\big) \;=\; 1x.
\end{aligned}$$

Example 2.4 In the Kleisli category $\mathcal{K}\ell(\mathcal{P})$ of the powerset monad $\mathcal{P}\colon \mathbf{Sets} \to \mathbf{Sets}$ the coproducts $+$ are also products (and thus "biproducts"). This means that we have a particularly simple example of a block structure, namely:

$$X \xrightarrow{\mathit{in}=\eta=\Delta} n \cdot X \xrightarrow{\mathit{out}=\varepsilon=\nabla} X \tag{3}$$

where η and ε are the unit and counit from Lemma 2.1. Explicitly, $\mathit{in}(x) = \{\kappa_1 x, \ldots, \kappa_n x\}$ and $\mathit{out}(\kappa_i x) = \{x\}$.

Example 2.5 The category **Hilb** of Hilbert spaces (over the complex numbers) also has biproducts $\oplus$, given by direct sums. Hence we can form blocks $\mathcal{B}_n(H) = n \cdot H = H \oplus \cdots \oplus H$ as before, for a Hilbert space H. But the obvious maps $in = \Delta$ and $out = \nabla$ as in (3) do not work in this case. One has to compensate by appropriate division. This can be done on either side, as in:

$$\begin{tikzcd}
H \arrow[r, "in=\frac{1}{n}\Delta"] \arrow[dr, equal] & n \cdot H \arrow[d, "out=\nabla=\sum"] & & H \arrow[r, "in=\Delta"] \arrow[dr, equal] & n \cdot H \arrow[d, "out=\frac{1}{n}\nabla"] \\
& H & & & H
\end{tikzcd} \tag{4}$$

where $(\frac{1}{n}\Delta)(x) = (\frac{1}{n}x, \ldots, \frac{1}{n}x)$ and $(\frac{1}{n}\nabla)(x_1, \ldots, x_n) = \frac{x_1 + \cdots + x_n}{n}$. Alternatively, it can be done in a more symmetric manner:

$$\begin{tikzcd}
H \arrow[r, "in=\frac{1}{\sqrt{n}}\Delta"] \arrow[dr, equal] & n \cdot H \arrow[d, "out=\frac{1}{\sqrt{n}}\nabla"] \\
& H
\end{tikzcd} \tag{5}$$

In this symmetric case we have $in^\dagger = out$, where $(-)^\dagger$ is the conjugate transpose. The equation $in^\dagger \circ in = \mathrm{id}$ makes in a dagger mono — and out a dagger epi.

3 Blocks and predicates

This section describes how predicates may be related to block structures via certain "characteristic" or "measurement" maps, much like in [10]. We assume that the predicates, on an object in a base category, carry the structure of an *effect algebra*. Such effect algebras are generalisations of logical structures used in classical logic (esp. Boolean algebras), in probabilistic logic (fuzzy predicates), and in quantum logic (projections and effects). Briefly, an effect algebra is a partial commutative monoid, with partial binary operation $\ovee$ and zero 0, together with a unique orthocomplent $x^\perp$, such that $x \ovee x^\perp = 1 = 0^\perp$, and such that $x \ovee 1$ is defined only for $x = 0$. The main example is the unit interval $[0, 1]$, with $r \ovee s$ defined and equal to the sum $r + s$ if $r + s \leq 1$, and with $r^\perp = 1 - r$. In a pointwise manner this structure extends to fuzzy predicates $[0, 1]^X$, see below. Each Boolean algebra also forms an effect algebra, with $x \ovee y$ defined and equal to the join $x \vee y$ if $x \wedge y = 0$. We shall use this below for powerset Boolean algebras $\mathcal{P}(X)$, where $\ovee$ is union of disjoint sets. For more information, see *e.g.* [5,4,12,13]. A morphism of effect algebras $f \colon E \to D$ is a function between the underlying sets satisfying $f(1) = 1$ and: if $x \perp y$, then $f(x) \perp f(y)$ and $f(x \ovee y) = f(x) \ovee f(y)$. This yields a category **EA**.

An *n-test* in an effect algebra E is an n-tuple $e = (e_1, \ldots, e_n)$ of elements $e_i \in E$ which satisfy $e_1 \ovee \cdots \ovee e_n = 1$. In this setting we describe a "logic of effects" categorically as a functor (or "indexed category") $Pred \colon \mathbf{A} \to \mathbf{EA}^{\mathrm{op}}$. It maps an object $X \in \mathbf{A}$ to the effect algebra $Pred(X)$ of predicates on X. A map $f \colon X \to Y$ gives rise to a "substitution" functor $Pred(f) \colon Pred(Y) \to Pred(X)$. In categorical logic it is often written as f^{-1}.

Definition 3.1 Let $\mathbf{A}$ be a category with an indexed category $\mathit{Pred}\colon \mathbf{A} \to \mathbf{EA}^{\mathrm{op}}$ of effect algebras, and with a block structure $\mathcal{B}_n\colon \mathbf{A} \to \mathbf{A}$. We say this is a *logical block structure* if

(i) for each $X \in \mathbf{A}$ and $n > 0$ there is a "universal" n-test on $\mathcal{B}_n(X)$, written as $\Omega = (\Omega_1, \dots, \Omega_n)$, with $\Omega_i \in \mathit{Pred}(\mathcal{B}_n(X))$ satisfying $\Omega_1 \ovee \cdots \ovee \Omega_n = 1$; moreover, these Ω_i should be stable under substitution, in the sense that $\mathcal{B}_n(f)^{-1}(\Omega_i) = \Omega_i$, for each $f\colon X \to Y$ in $\mathbf{A}$;

(ii) for each $X \in \mathbf{A}$ and n-test $p = (p_1, \dots, p_n)$ on X, where $p_i \in \mathit{Pred}(X)$ satisfy $p_1 \ovee \cdots \ovee p_n = 1$, there is a "characteristic" map $\mathit{char}_p\colon X \to \mathcal{B}_n(X)$ in $\mathbf{A}$ with $\mathit{char}_p^{-1}(\Omega_i) = p_i$, for each $i \in n$.

The characteristic map yields a block opening $\mathit{char}_p(x) \in \mathcal{B}_n(X)$ whose n different options are determined by the n predicates p_i in p.

Our first example clearly shows the importance of understanding powersets of predicates as effect algebras, because the *disjoint* union is crucial for having characteristic maps.

Example 3.2 On the Kleisli category $\mathcal{K}\ell(\mathcal{P})$ of the powerset monad $\mathcal{P}$ there is an indexed category $\mathit{Pred}\colon \mathcal{K}\ell(\mathcal{P}) \to \mathbf{EA}^{\mathrm{op}}$ given by ordinary predicates: $\mathit{Pred}(X) = \mathcal{P}(X)$. This set of predicates is a Boolean algebra, and thus an effect algebra, with sum $\ovee$ defined as union, but only for disjoint subsets. For a Kleisli map $f\colon X \to Y$ we have a substitution functor:

$$\mathcal{P}(Y) \xrightarrow{f^{-1} = \mathit{Pred}(f)} \mathcal{P}(X) \qquad \text{given by} \qquad V \longmapsto \{x \mid f(x) \subseteq V\}.$$

(This substitution f^{-1} is not the same as inverse image, which is often also written as f^{-1}.)

We show that the block structure $\mathcal{B}_n(X) = n \cdot X$ from Example 2.4 is a logical block structure. For each number $n > 0$ and set X there is an n-test $\Omega = (\Omega_1, \dots, \Omega_n)$ on $\mathcal{B}_n(X) = n \cdot X$ given by subsets:

$$\Omega_i = \{\kappa_i x \mid x \in X\} \subseteq n \cdot X = X + \cdots + X = \mathcal{B}_n(X).$$

These subsets Ω_i are all disjoint, so their effect algebra sum $\Omega_1 \ovee \cdots \ovee \Omega_n$ exists and equals the maximal predicate $1 = n \cdot X \subseteq n \cdot X$ in $\mathit{Pred}(n \cdot X)$. It is easy to see that Ω is stable under composition: for $f\colon X \to Y$ in $\mathcal{K}\ell(\mathcal{P})$,

$$\mathcal{B}_n(f)^{-1}(\Omega_i) = \{z \in n \cdot X \mid (n \cdot f)(z) \subseteq \Omega_i\} = \{\kappa_i x \mid x \in X\} = \Omega_i.$$

For an arbitrary n-test $U = (U_1, \dots, U_n)$ on X, where $U_1 \ovee \cdots \ovee U_n = 1$, there is a characteristic map in the Kleisli category $\mathcal{K}\ell(\mathcal{P})$:

$$X \xrightarrow{\mathit{char}_U} \mathcal{B}_n(X) \qquad \text{namely} \qquad x \longmapsto \{\kappa_i x\}, \quad \text{if } x \in U_i.$$

Since the predicates U_i are mutually disjoint with join X, this forms a well-defined map. The required substitution equation in Definition 3.1 (2) holds:

$$\mathit{char}_U^{-1}(\Omega_i) = \{x \mid \mathit{char}_U(x) \subseteq \Omega_i\} = \{x \mid x \in U_i\} = U_i.$$

It is not hard to verify that this map $char_U\colon X \to \mathcal{B}_n(X)$ is an Eilenberg-Moore coalgebra of the comonad $\mathcal{B}_n = n \cdot (-)$, *i.e.* that the equations $out_n \odot char_U = \mathrm{id}$ and $\delta \odot char_U = \mathcal{B}_n(char_U) \odot char_U$ hold, where δ is the comultiplication from Lemma 2.1. In fact one can prove that there is a bijective correspondence:

$$\frac{\text{Boolean } n\text{-tests } U = (U_1, \ldots, U_n) \text{ in } \mathcal{P}(X)}{\overline{\text{Eilenberg-Moore coalgebras } X \longrightarrow \mathcal{B}_n(X) \text{ in } \mathcal{K}\ell(\mathcal{P})}} \tag{6}$$

With intersection $\cap$ as multiplication operation, each of these predicates U_i is a projection, since $U_i^2 = U_i \cap U_i = U_i$.

Example 3.3 The Kleisli category $\mathcal{K}\ell(\mathcal{D})$ carries an indexed category $Pred\colon \mathcal{K}\ell(\mathcal{D}) \to \mathbf{EA}^{\mathrm{op}}$ given by fuzzy predicates: $Pred(X) = [0,1]^X$. The effect algebra structure on $[0,1]^X$ is inherited pointwise from $[0,1]$. In particular, for $p, q \in [0,1]^X$, if $p(x) + q(x) \leq 1$ for all $x \in X$, then $p \ovee q$ is defined and $(p \ovee q)(x) = p(x) + q(x)$. Each map $f\colon X \to Y$ in $\mathcal{K}\ell(\mathcal{D})$ yields a substitution functor:

$$[0,1]^Y \xrightarrow{f^{-1} = Pred(f)} [0,1]^X \qquad \text{by} \qquad q \longmapsto \lambda x.\, \textstyle\sum_y f(x)(y) \cdot q(y).$$

We show that the copower block structure $X \mapsto n \cdot X$ from Example 2.3 is logical.

(i) The "universal" n-test Ω consists of predicates $\Omega_i \in [0,1]^{n \cdot X}$, given by $\Omega_i(\kappa_j x)$ is 1 if $i = j$ and 0 otherwise. Then: $\Omega_1 \ovee \cdots \ovee \Omega_n = 1$; moreover these predicates Ω_i are stable under substitution.

(ii) For an arbitrary n-test p on X, given by $p_i \in [0,1]^X$ with $p_1 \ovee \cdots \ovee p_n = 1$, we define a characteristic map $char_p\colon X \to n \cdot X$ in $\mathcal{K}\ell(\mathcal{D})$ via the convex sums:

$$char_p(x) = p_1(x)\kappa_1 x + \cdots + p_n(x)\kappa_n x,$$

using that $p_1(x) + \cdots + p_n(x) = 1$. Then:

$$\begin{aligned} char_p^{-1}(\Omega_i)(x) &= \textstyle\sum_{z \in n \cdot X} char_p(x)(z) \cdot \Omega_i(z) \\ &= \textstyle\sum_{y \in X} char_p(x)(\kappa_i y) \\ &= p_i(x). \end{aligned}$$

In general the map $char_p\colon X \to \mathcal{B}_n(X)$ does not form an Eilenberg-Moore coalgebra of the comonad $\mathcal{B}_n$: we do have $out_n \odot char_p = \mathrm{id}$, but the δ-law may fail. However, the law holds for n-tests given by fuzzy projections $p_i \in [0,1]^X$, satisfying $p_i^2 = p_i$. Automatically, $p_i p_j = 0$, for $j \neq i$, since the p_i add up to 1. It is not hard to see that these projections correspond to "Boolean" fuzzy tests, determined by indicator functions $\mathbf{1}_{U_i}\colon X \to [0,1]$ with $\mathbf{1}_{U_i}(x) = 1$ if $x \in U_i$ and $\mathbf{1}_{U_i}(x) = 0$ otherwise, for disjoint subsets $U_i \subseteq X$ with $U_1 \ovee \cdots \ovee U_n = 1$. Thus we have a bijective correspondence like in (6):

$$\begin{array}{c} \text{Boolean } n\text{-tests } U = (U_1, \ldots, U_n) \text{ in } \mathcal{P}(X) \\ \hline\hline n\text{-tests of projections } p = (p_1, \ldots, p_n) \text{ in } [0,1]^X \text{ with } p_i^2 = p_i \\ \hline\hline \text{Eilenberg-Moore coalgebras } X \longrightarrow \mathcal{B}_n(X) \text{ in } \mathcal{K}\ell(\mathcal{D}) \end{array} \tag{7}$$

The fuzzy predicates in $[0,1]^X$ form not only an effect algebra but an *effect module* (see [13] for details): they come with scalar multiplication, with scalars r from $[0,1]$, via $r \cdot p = \lambda x.\, r \cdot p(x)$. In the subcategory $\mathbf{EMod} \hookrightarrow \mathbf{EA}$ of such effect modules maps preserve the scalar multiplication. In this setting there are alternative characterisations of n-tests in $[0,1]^X$, in the style of [10], which we express via bijective correspondences:

$$\begin{array}{c} \text{fuzzy } n\text{-tests } p = (p_1, \ldots, p_n) \text{ in } [0,1]^X \\ \hline\hline \text{effect module maps } [0,1]^n \longrightarrow [0,1]^X \\ \hline\hline \text{Kleisli maps } X \longrightarrow \mathcal{B}_n(1) \text{ in } \mathcal{K}\ell(\mathcal{D}) \\ \hline\hline \text{Kleisli maps } X \xrightarrow[f]{} \mathcal{B}_n(X) \text{ with } \mathit{out}_n \odot f = \mathrm{id} \end{array} \tag{8}$$

where $1 = \{*\}$ is the singleton set. The first correspondence is standard. An n-test p corresponds to a Kleisli map $g \colon X \to \mathcal{B}_n(1)$ via $g(x) = \sum_i p_i(x)\kappa_i *$, and to a map $f \colon X \to \mathcal{B}_n(X)$ via $f = \mathit{char}_p$. Such a map f gives rise to an n-test with predicates $p_i = \lambda x.\, f(x)(\kappa_i x)$.

Example 3.4 The distribution monad $\mathcal{D}$ is used in a categorical approach to *discrete* probability. For the continuous case one uses the Giry monad [8]. It is defined as monad $\mathcal{G} \colon \mathbf{Meas} \to \mathbf{Meas}$ on the category of measurable spaces, where $\mathcal{G}(X)$ contains the probability measures $\Sigma_X \to [0,1]$, defined on the measurable subsets $\Sigma_X \subseteq \mathcal{P}(X)$. We briefly illustrate how it carries a logical block structure, in line with [11]. We follow the constructions and notation used there.

The logic is given by a functor $\mathit{Pred} \colon \mathcal{K}\ell(\mathcal{G}) \to \mathbf{EMod}^{\mathrm{op}}$ that sends a measurable space X to the homset $\mathit{Pred}(X) = \mathbf{Meas}(X, [0,1])$ of measurable maps to $[0,1]$, with pointwise effect module structure. For a Kleisli map $f \colon X \to \mathcal{G}(Y)$ and predicate $q \colon Y \to [0,1]$ one defines substitution by integration:

$$\mathit{Pred}(f)(q) \;=\; f^{-1}(q) \;=\; \lambda x.\, \int q \,\mathrm{d}f(x).$$

There is a (comonad) block structure $\mathcal{B}_n(X) = n \cdot X$ defined via copowers on $\mathcal{K}\ell(\mathcal{G})$, with the $\mathit{in} \colon X \to \mathcal{G}(n \cdot X)$ and $\mathit{out} \colon n \cdot X \to \mathcal{G}(X)$ maps given by:

$$\mathit{in}(x) \;=\; \lambda M \in \Sigma_{n \cdot X}.\, \tfrac{1}{n} \textstyle\sum_i \mathbf{1}_M(\kappa_i x) \qquad\qquad \mathit{out}(\kappa_i x) \;=\; \lambda N \in \Sigma_X.\, \mathbf{1}_N(x).$$

The predicates $\Omega_i \colon n \cdot X \to [0,1]$ are defined, as in Example 3.3, as $\Omega_i = \mathbf{1}_{\kappa_i X}$, *i.e.* as $\Omega_i(\kappa_j x) = 1$ if $i = j$ and $\Omega_i(\kappa_j x) = 0$ otherwise. And for an n-test $p = (p_1, \ldots, p_n)$ of predicates $p_i \in \mathit{Pred}(X)$ with $p_1 \ovee \cdots \ovee p_n = 1$, we can define a characteristic map $\mathit{char}_p \colon X \to \mathcal{B}_n(X)$ in $\mathcal{K}\ell(\mathcal{G})$ by:

$$\mathit{char}_p(x) \;=\; \lambda M \in \Sigma_{n \cdot X}.\, \textstyle\sum_i \, p_i(x) \cdot \mathbf{1}_M(\kappa_i x).$$

Also in this case the n-tests $p_i \colon X \to [0,1]$ that consist of projections, *i.e.* that satisfy $p_i^2 = p_i$, can be characterised as Eilenberg-Moore coalgebras, like in (7). They also correspond to indicator functions $\mathbf{1}_{M_i}$, for $M_i \in \Sigma_X$ pairwise disjoint.

Since the measurable subsets Σ_X form an effect algebra, with $\ovee$ given by disjoint union, they form n-tests in Σ_X. Thus we get bijective correspondences:

$$\begin{array}{c} n\text{-tests } M = (M_1, \ldots, M_n) \text{ in } \Sigma_X \\ \hline\hline n\text{-tests of projections } p = (p_1, \ldots, p_n) \text{ in } \mathbf{Meas}(X, [0,1]) \text{ with } p_i^2 = p_i \\ \hline\hline \text{Eilenberg-Moore coalgebras } X \longrightarrow \mathcal{B}_n(X) \text{ in } \mathcal{K}\ell(\mathcal{G}) \end{array} \tag{9}$$

Example 3.5 In the context of Hilbert spaces, several of the ingredients encountered above are present, but we do *not* find a logical block structure, for the standard logic of effects. We briefly describe the situation, building on Example 2.5.

We start with the logic. We write $\mathbf{Hilb}_{\mathrm{isom}} \hookrightarrow \mathbf{Hilb}$ for the subcategory of Hilbert spaces with isometries between them. Such an isometry f is bounded linear function that is a "dagger mono", *i.e.* satisfies $f^\dagger \circ f = \mathrm{id}$. There is an "effect" predicate functor $\mathcal{E}f \colon \mathbf{Hilb}_{\mathrm{isom}} \to \mathbf{EMod}^{\mathrm{op}}$ that sends a Hilbert space H to the set of effects:

$$\mathcal{E}f(H) = \{A \colon H \to H \mid 0 \leq A \leq \mathrm{id}\}.$$

These effects are the quantum fuzzy/unsharp predicates, see *e.g.* [16,15,5]. An effect $A \ovee B$ is defined and equal to $A+B$ if $A+B \leq \mathrm{id}$. The orthocomplement is given by $A^\perp = \mathrm{id} - A$. Scalar multiplication rA, for $r \in [0,1]$ is done in a pointwise manner. Hence this $\mathcal{E}f(H)$ is an effect module.

For a dagger monic map $f \colon H \rightarrowtail K$ one defines $f^{-1} = \mathcal{E}f(f) = f^\dagger \circ (-) \circ f \colon \mathcal{E}f(K) \to \mathcal{E}f(H)$. This substitution functor f^{-1} preserves the effect module structure because f is a dagger mono.

Let $\mathcal{B}_n(H) = n \cdot H = H \oplus \cdots \oplus H$ be the block structure on **Hilb** from Example 2.5. There is an n-test $\Omega = (\Omega_1, \ldots, \Omega_n)$ of effects $\Omega_i = \kappa_i \circ \pi_i \in \mathcal{E}f(\mathcal{B}_n(H))$. More explicitly, $\Omega_i(x_1, \ldots, x_n) = (0, \ldots, 0, x_i, 0, \ldots, 0)$. These Ω_i's are stable under substitution.

For an n-test $A = (A_1, \ldots, A_n)$ of effects $A_i \in \mathcal{E}f(H)$ we can define a characteristic map $\mathit{char}_A \colon H \to \mathcal{B}_n(H)$ in $\mathbf{Hilb}_{\mathrm{isom}}$ as n-tuple of square roots of (positive) maps:

$$\mathit{char}_A = \langle \sqrt{A_1}, \ldots, \sqrt{A_n} \rangle \colon H \longrightarrow H \oplus \cdots \oplus H = \mathcal{B}_n(H).$$

This characteristic map is a dagger mono, since, as shown in [10]:

$$\big(\mathit{char}_A\big)^\dagger \circ \mathit{char}_A = [\sqrt{A_1}, \ldots, \sqrt{A_n}] \circ \langle \sqrt{A_1}, \ldots, \sqrt{A_n} \rangle = A_1 + \cdots + A_n = \mathrm{id}.$$

Clearly, we have:

$$\begin{aligned} (\mathit{char}_A)^{-1}(\Omega_i) &= \big(\mathit{char}_A\big)^\dagger \circ \Omega_i \circ \mathit{char}_A \\ &= [\sqrt{A_1}, \ldots, \sqrt{A_n}] \circ \kappa_1 \circ \pi_i \circ \langle \sqrt{A_1}, \ldots, \sqrt{A_n} \rangle \\ &= \sqrt{A_i} \circ \sqrt{A_i} \\ &= A_i. \end{aligned}$$

The map $\mathit{in} = \frac{1}{\sqrt{n}}\Delta \colon H \to \mathcal{B}_n(H)$ in (5) arises in this manner as characteristic map of the n-test $(\frac{1}{n}\mathrm{id}, \ldots, \frac{1}{n}\mathrm{id})$. However, the corresponding "out" map in (5),

$\mathit{out} = \mathit{in}^{\dagger} = \frac{1}{\sqrt{n}}\nabla$, is *not* a morphism in the category $\mathbf{Hilb}_{\mathrm{isom}}$, since it is not a dagger mono (but a dagger epi). Thus this does *not* give us a logical block structure in $\mathbf{Hilb}_{\mathrm{isom}}$.

Using $\oplus$ as coproduct we have a comonad structure (ε, δ) on $\mathcal{B}_n = n \cdot (-)$, as in Lemma 2.1. Following [3] we call a map $c\colon H \to \mathcal{B}_n(H)$ in **Hilb** *self-adjoint* if the following diagram commutes.

$$\begin{tikzcd}
H \arrow[r, "c"] \arrow[d, "\eta=\Delta"'] & \mathcal{B}_n(H) \\
\mathcal{B}_n(H) \arrow[r, "\delta=\kappa_1\oplus\cdots\oplus\kappa_n"'] & \mathcal{B}_n(\mathcal{B}_n(H)) \arrow[u, "\mathcal{B}_n(c^{\dagger})"']
\end{tikzcd}$$

This means that each component $c_i = \pi_i \circ c\colon H \to H$ is self-adjoint, *i.e.* satisfies $c_i^{\dagger} = c_i$.

The subset of projections (or sharp predicates) $\mathcal{P}r(H) \hookrightarrow \mathcal{E}f(H)$ contains those $p\colon H \to H$ with $p \circ p = p = p^{\dagger}$. An n-test $A = (A_1, \ldots, A_n)$ in $\mathcal{E}f(H)$ is called a *von Neumann* test if each A_i satisfies $A_i \circ A_i = A_i$ and $A_i \circ A_j = 0$ for each $j \neq i$. Such an A_i is then a projection. One of the main results of [3] (specifically: Thm. 16.6) says that there is a bijective correspondence:

$$\frac{\text{von Neumann } n\text{-tests } p = (p_1, \ldots, p_n) \text{ in } \mathcal{E}f(H)}{\text{self-adjoint Eilenberg-Moore coalgebras } H \to \mathcal{B}_n(H)} \qquad (10)$$

A test p corresponds to its characteristic map $\mathit{char}_p = \langle \sqrt{p_1}, \ldots, \sqrt{p_n} \rangle = \langle p_1, \ldots, p_n \rangle$.

4 Copower block structure on C^*-algebras

In the present context all C^*-algebras have a unit. The maps $f\colon A \to B$ between C^*-algebras that we consider are linear functions which are unital (preserve the unit) and positive (preserve positive elements: for each $x \in A$ there is an $y \in B$ with $f(x^*x) = y^*y$). We often refer to these morphisms as 'PU-maps'. We shall write $\mathbf{Cstar}_{\mathrm{PU}}$ for the category of C^*-algebras with such unital positive maps, and $\mathbf{CCstar}_{\mathrm{PU}} \hookrightarrow \mathbf{Cstar}_{\mathrm{PU}}$ for the full subcategory of C^*-algebras with commutative multiplication. These categories of C^*-algebras are most naturally used in opposite form — as $(\mathbf{Cstar}_{\mathrm{PU}})^{\mathrm{op}}$ and $(\mathbf{CCstar}_{\mathrm{PU}})^{\mathrm{op}}$ — just like the category **cHA** of complete Heyting algebras typically occurs in opposite form, as category of locales $\mathbf{Loc} = \mathbf{cHA}^{\mathrm{op}}$, see *e.g.* [14].

In the literature on C^*-algebras it is most common to use *-homomorphism as maps. These preserve multiplication (M), involution (I) and are unital (U). In [6] these *-homomorphisms are called MIU-maps, in order to distinguish them from the PU-maps which are used here. MIU-maps are very restrictive, which is useful for Gelfand duality. But the PU-maps are the appropriate notion in a probabilistic or quantum context.

Let's write $\mathcal{K}\ell_{\mathbb{N}}(\mathcal{D}) \hookrightarrow \mathcal{K}\ell(\mathcal{D})$ for the full subcategory with natural numbers $n \in \mathbb{N}$ as objects, considered as n-element set. There is a full and faithful functor $\mathcal{K}\ell_{\mathbb{N}}(\mathcal{D}) \to (\mathbf{CCstar}_{\mathrm{PU}})^{\mathrm{op}}$, which sends an object n to $\mathbb{C}^n = \mathbb{C} \times \cdots \times \mathbb{C}$, the

n-fold power of the complex numbers $\mathbb{C}$; it sends a Kleisli map $f: n \to m$ to the PU-map $\mathbb{C}^m \to \mathbb{C}^n$ given by $v \mapsto \lambda i \in n.\, \sum_{j \in m} f(i)(j) \cdot v(j)$. This functor restricts to an equivalence between $\mathcal{K}\ell_{\mathbb{N}}(\mathcal{D})$ and the subcategory of finite dimensional commutative C^*-algebras, see [6]. In fact, in [6] it is shown that there is an equivalence between $(\mathbf{CCstar}_{\mathrm{PU}})^{\mathrm{op}}$ and the Kleisli category of the "Radon" monad on the compact Hausdorff spaces. The point we are trying to make is that the category $(\mathbf{CCstar}_{\mathrm{PU}})^{\mathrm{op}}$ of *commutative* C^*-algebras is a natural universe for probabilistic (monadic) computation.

In general, the multiplication term ab, for two positive elements a, b in a C^*-algebra, need not be positive. The following easy observations will be useful.

Lemma 4.1 *Let a be a positive element in an arbitrary C^*-algebra. Then:*

(i) *x^*ax is positive, for each element x;*

(ii) *xax is positive, for each* positive *x;*

Proof Write $a = y^*y$; then $x^*ax = x^*y^*yx = (yx)^*(yx)$ is clearly positive. If x is positive itself, then $x^* = x$, so the second point follows from the first one. □

For two C^*-algebras A, B we write $A \oplus B$ for the C^*-algebra with product $A \times B$ as underlying set, with componentwise operations, and with maximum of the norms. Together with the usual projection and pairing operations this $\oplus$ forms a product in $\mathbf{Cstar}_{\mathrm{PU}}$ and $\mathbf{CCstar}_{\mathrm{PU}}$, and thus a coproduct in their dual categories. By Lemma 2.1 the mapping

$$A \longmapsto \mathcal{B}_n(A) \overset{\text{def}}{=} n \cdot A = A \oplus \cdots \oplus A$$

is a comonad on $(\mathbf{Cstar}_{\mathrm{PU}})^{\mathrm{op}}$ and $(\mathbf{CCstar}_{\mathrm{PU}})^{\mathrm{op}}$. We show that it extends to a block structure, both on $(\mathbf{Cstar}_{\mathrm{PU}})^{\mathrm{op}}$ and $(\mathbf{CCstar}_{\mathrm{PU}})^{\mathrm{op}}$, namely:

$$\begin{tikzcd} A \arrow[r, "in_n"] & \mathcal{B}_n(A) \arrow[r, "out_n"] & A, \end{tikzcd} \tag{11}$$

where out_n is the diagonal (counit) map $A \to A^n$ given by $out_n(a) = (a, \ldots, a)$. The map $in_n : A^n \to A$ takes the average: $in_n(a_1, \ldots, a_n) = \frac{a_1 + \cdots + a_n}{n}$. Keeping the 'opposite' in mind we see that the required block structure equation holds:

$$\big(out_n \circ^{\mathrm{op}} in_n\big)(a) = in_n\big(out_n(a)\big) = \frac{a + \cdots + a}{n} = a.$$

We further notice that in_n is a PU-map, but out_n is a MIU-map.

For a C^*-algebra A we write $[0,1]_A = \{x \in A \mid 0 \le x \le 1\}$ for the "effects" in A, that is, for the positive elements below the unit. These form an effect algebra, with $x \ovee y$ defined and equal to $x + y$ if $x + y \le 1$. The orthocomplement of $x \in [0,1]_A$ is $1 - x$. In fact, $[0,1]_A$ is not just an effect algebra but an effect module, since scalar multiplication rx, where r is in the unit interval $[0,1] \subseteq \mathbb{R}$ and $x \in [0,1]_A$, yields an effect $rx \in [0,1]_A$. Each PU-map $f: A \to B$ restricts to a map of effect algebras $[0,1]_A \to [0,1]_B$. In [6] it is shown that the mapping $A \mapsto [0,1]_A$ yields a full and faithful functor $\mathbf{Cstar}_{\mathrm{PU}} \to \mathbf{EMod}$. We shall use it as $\mathit{Pred}: (\mathbf{Cstar}_{\mathrm{PU}})^{\mathrm{op}} \to \mathbf{EMod}^{\mathrm{op}}$, where $\mathit{Pred}(A) = [0,1]_A$. The substitution functor $Pred(f) = f^{-1}\colon [0,1]_B \to [0,1]_A$ associated with $f\colon A \to B$ in $(\mathbf{Cstar}_{\mathrm{PU}})^{\mathrm{op}}$ is obtained simply by restriction.

In this situation, like in (8), tests can be characterised in various ways.

Lemma 4.2 *For a C^*-algebra A, there are bijective correspondences between:*

$$\begin{array}{c} n\text{-}\mathit{tests}\ p = (p_1, \dots, p_n)\ \mathit{in}\ [0,1]_A \\ \hline\hline \mathit{effect\ module\ maps}\ [0,1]^n \longrightarrow [0,1]_A \\ \hline\hline \mathit{maps}\ A \longrightarrow \mathcal{B}_n(\mathbb{C})\ \mathit{in}\ (\boldsymbol{Cstar}_{PU})^{op} \end{array} \tag{12}$$

Proof Given an n-test $e = (e_1, \dots, e_n)$, define $h\colon A \to \mathcal{B}_n(\mathbb{C})$ in $(\mathbf{Cstar}_{\mathrm{PU}})^{\mathrm{op}}$, that is $h\colon \mathbb{C}^n \to A$ in $\mathbf{Cstar}_{\mathrm{PU}}$, by $h(z_1, \dots, z_n) = \sum_i z_i e_i$. This h is clearly positive, and unital since $h(1, \dots, 1) = \sum_i e_i = 1$. Conversely, a PU-map $f\colon \mathbb{C}^n \to A$ is determined by the values $f(|\,i\,\rangle)$, where $|\,i\,\rangle$ is the standard base vector $(0, \dots, 1, \dots, 0) \in \mathbb{C}^n$. Since $0 \leq |\,i\,\rangle \leq 1$ one has $f(|\,i\,\rangle) \in [0,1]_A$. □

Proposition 4.3 *Let $\overrightarrow{a_i} = (a_1, \dots, a_n) \in A^n = \mathcal{B}_n(A)$ be an n-tuple in a C^*-algebra A.*

(i) *If $\sum_i a_i^* a_i = 1$ there is a PU-map:*

$$\mathcal{B}_n(A) \xrightarrow{meas(\overrightarrow{a_i})} A \quad \mathit{given\ by} \quad \overrightarrow{b_i} \longmapsto \textstyle\sum_i a_i^* b_i a_i$$

(ii) *If $a_i^* a_i = 1$ for each i, then there is a PU-map:*

$$\mathcal{B}_n(A) \xrightarrow{map(\overrightarrow{a_i}) = \prod_i a_i^*(-)a_i} \mathcal{B}_n(A) \quad \mathit{given\ by} \quad \overrightarrow{b_i} \longmapsto (a_1^* b_1 a_1, \dots, a_n^* b_n a_n)$$

Proof The conditions $\sum_i a_i^* a_i = 1$ and $\forall i.\, a_i^* a_i = 1$ ensure that the functions $meas(\overrightarrow{a_i})$ and $map(\overrightarrow{a_i})$ are unital. Positivity is trivial, by Lemma 4.1. □

Example 4.4 Let $\psi = (z_1, \dots, z_n) \in \mathbb{C}^n$ be a state, so that $\|\psi\| = 1$. This means that $\langle \psi \,|\, \psi \rangle = \sum_i \overline{z_i} z_i = \sum_i |z_i|^2 = 1$, where $\overline{\cdot}$ is conjugation of complex numbers. Hence in each C^*-algebra A this ψ gives rise to an n-tuple $z_i 1$ with $\sum_i (z_i 1)^*(z_i 1) = 1$. These elements $z_i 1 \in A$ arise via the unique map $\mathbb{C} \to A$, using initiality of $\mathbb{C}$ among C^*-algebras. The "measure" map from Proposition 4.3 then gives a PU-map $\mathcal{B}_n(A) \to A$, namely

$$(b_1, \dots, b_n) \longmapsto \textstyle\sum_i (z_i 1)^* b_i (z_i 1) = \sum_i (\overline{z_i} z_i) b_i = \sum_i |z_i|^2 b_i.$$

In the opposite category this operation forms a map $A \to \mathcal{B}_n(A)$ which describes how a context is opened and initialised by the state $\psi \in \mathbb{C}^n$, via a probabilistic mixture determined by $|z_i|^2 \in [0,1]$, corresponding to the Born rule.

It turns out that the "copower" definition $\mathcal{B}_n = n \cdot (-)$ yields a logical block structure, also for C^*-algebras, with predicate logic given by their effects: $Pred(A) = [0,1]_A$. In the next section we show that there is another block structure.

Proposition 4.5 *The assignment $A \mapsto \mathcal{B}_n(A) = A \oplus \cdots \oplus A$, with maps (11), is a logical block structure, both on $(\boldsymbol{Cstar}_{PU})^{op}$ and on $(\boldsymbol{CCstar}_{PU})^{op}$.*

(i) *The universal n-test $\Omega = (\Omega_1, \ldots, \Omega_n)$ consists of $\Omega_i \in [0,1]_{\mathcal{B}_n(A)} = ([0,1]_A)^n$ given by the n-tuple of effects $(0, \ldots, 1, \ldots, 0)$, with 1 only at the i-th position.*

(ii) *For an n-test $e = (e_1, \ldots, e_n)$ one can define a characteristic maps $char_e \colon A \to \mathcal{B}_n(A)$ as:*

$$char_e(a_1, \ldots, a_n) = \sqrt{e_1}a_1\sqrt{e_1} + \cdots + \sqrt{e_n}a_n\sqrt{e_n}.$$

If A is commutative, we get $char_e(a_1, \ldots, a_n) = \sum_i e_i a_i$.

Proof It is clear that the predicates Ω_i are stable under substitution. Further, $char_e^{-1}(\Omega_i) = char_e(0, \ldots, 1, \ldots, 0) = \sqrt{e_i}\sqrt{e_i} = e_i$. □

The following result gives a C^*-algebraic version of the correspondences (6), (7), and (10). It only applies in the commutative case.

Generalising Example 3.5 we call an n-test $e = (e_1, \ldots, e_n)$ in a C^*-algebra a *von Neumann* n-test if each e_i is a projection, *i.e.* satisfies $e_i^2 = e_i$, and satisfies $e_i e_j = 0$ for each $j \neq i$.

Theorem 4.6 *In a C^*-algebra A there are bijective correspondences:*

$$\begin{array}{c}
\textit{von Neumann } n\textit{-tests } e = (e_1, \ldots, e_n) \textit{ in } [0,1]_A \\
\hline\hline
\textit{maps } A \dashrightarrow \mathcal{B}_n(\mathbb{C}) \textit{ in } (\boldsymbol{Cstar}_{MIU})^{op} \\
\hline\hline
\textit{Eilenberg-Moore coalgebras } A \longrightarrow \mathcal{B}_n(A) \textit{ in } (\boldsymbol{Cstar}_{PU})^{op}
\end{array} \ (*) \tag{13}$$

where the second correspondence, marked with $(*)$, only works if the C^*-algebra A is commutative.

Proof We first do the first correspondence. Given a von Neumann n-test $e = (e_1, \ldots, e_n)$ we can define a MIU-map $f \colon \mathbb{C}^n \to A$ as sum of scalar multiplications: $f(z_1, \ldots, z_n) = \sum_i z_i e_i$, as in the proof of Lemma 4.2. It now preserves multiplication:

$$\begin{aligned}
f(\overrightarrow{z_i}) \cdot f(\overrightarrow{w_i}) \;=\; \big(\textstyle\sum_i z_i e_i\big) \cdot \big(\textstyle\sum_i w_i e_i\big) &= \textstyle\sum_{i,j} z_i e_i \cdot w_j e_j \\
&= \textstyle\sum_{i,j} (z_i \cdot w_j)(e_i \cdot e_j) \\
&= \textstyle\sum_i (z_i \cdot w_i) e_i \;=\; f(\overrightarrow{(z \cdot w)_i}).
\end{aligned}$$

In the other direction, given such a MIU-map $f \colon \mathbb{C}^n \to A$ we obtain an n-test of effects $e_i = f(|\,i\,\rangle)$, like before. Now we have:

$$e_i e_j \;=\; f(|\,i\,\rangle) f(|\,j\,\rangle) \;=\; f(|\,i\,\rangle|\,j\,\rangle) \;=\; \begin{cases} f(|\,i\,\rangle) = e_i & \text{if } i = j \\ f(0) = 0 & \text{otherwise.} \end{cases}$$

Hence the e_i form mutually orthogonal projections, and thus a von Neumann test.

For the second correspondence, assume A is commutative. Let $e = (e_1, \ldots, e_n)$ be a von Neumann n-test. The corresponding characteristic PU-map $\mathcal{B}_n(A) \to A$ from Proposition 4.5, is given by $char_e(\overrightarrow{a}) = \sum_i \sqrt{e_i}a_i\sqrt{e_i} = \sum_i e_i a_i$. The latter simple form, resulting from commutativity, is crucial for proving the ε-equation for a coalgebra, in the opposite category $(\mathbf{Cstar}_{\mathrm{PU}})^{\mathrm{op}}$.

$$\begin{aligned}
\big(\varepsilon \circ^{\mathrm{op}} \mathit{char}_e\big)(a) &= \big(\mathit{char}_e \circ \mathit{out}\big)(a) \\
&= \mathit{char}_e(a, \ldots, a) \\
&= \textstyle\sum_i e_i a \\
&= (\textstyle\sum_i e_i)a \\
&= 1a \\
&= a. \\
\big(\mathcal{B}_n(\mathit{char}_e) \circ^{\mathrm{op}} \mathit{char}_e\big)(\overrightarrow{t_i}) &= \mathit{char}_e\big(\mathit{char}_e(t_1), \ldots, \mathit{char}_e(t_n)\big) \quad \text{for } t_i \in A^n \\
&= \textstyle\sum_i e_i(\sum_j e_j t_{ij}) \\
&= \textstyle\sum_{i,j} e_i e_j t_{ij} \\
&= \textstyle\sum_i e_i t_{ii} \\
&= \mathit{char}_e\big(\delta(\overrightarrow{t_i})\big) \\
&= \big(\delta \circ^{\mathrm{op}} \mathit{char}_e\big)(\overrightarrow{t_i}).
\end{aligned}$$

Finally, assuming a coalgebra $f \colon A \to \mathcal{B}_n(A)$ in $(\mathbf{Cstar}_{\mathrm{PU}})^{\mathrm{op}}$, we define effects $e_i = f(|\,i\,\rangle_A)$, where $|\,i\,\rangle_A = (0, \ldots, 1, \ldots, 0) \in ([0,1]_A)^n = [0,1]_{A^n} = [0,1]_{\mathcal{B}_n(A)}$. Clearly, $e_1 \ovee \cdots \ovee e_1 = f(1) = 1$. The equation $\varepsilon \circ^{\mathrm{op}} f = f \circ \mathit{out} = \mathrm{id}$ yields that f is a "map of bimodules":

$$b \cdot f(a_1, \ldots, a_n) \cdot c = f\big(\mathit{out}(b) \cdot (a_1, \ldots, a_n) \cdot \mathit{out}(c)\big) \tag{14}$$

This follows from [22, Thm. 1], because f is a PU-map and out a MIU-map, and will be used without further ado.

We can now prove that the e_i are mutually orthogonal projections. Consider the "matrix" $t = |\,j\,\rangle\langle\,i\,| \in \mathcal{B}_n(\mathcal{B}_n(A))$, so that $t(x)(y)$ is 1 if $x = i$ and $y = j$, and 0 otherwise. Then:

$$\begin{aligned}
e_i \cdot e_j \;=\; f(|\,i\,\rangle) \cdot e_j &= f\big(|\,i\,\rangle \cdot \mathit{out}(e_j)\big) && \text{by (14)} \\
&= f\big(0, \ldots, 0, e_j, 0, \ldots, 0\big) && \text{with } e_j \text{ at position } i \\
&= f\big(f(0), \ldots, f(|\,j\,\rangle), \ldots, f(0)\big) \\
&= \big(\mathcal{B}_n(f) \circ^{\mathrm{op}} f\big)(t) \\
&= \big(\delta \circ^{\mathrm{op}} f\big)(t) \\
&= f(\lambda x.\, t(x)(x)) \\
&= \begin{cases} f(|\,i\,\rangle) = e_i & \text{if } i = j \\ f(0) = 0 & \text{otherwise.} \end{cases}
\end{aligned}$$

□

We conclude this section with some basic observations. First, the opening of a the block via $\mathit{in}_n \colon A \to \mathcal{B}_n(A)$ as in (11) can be described via the characteristic maps $A \to \mathcal{B}_n(A)$, for the "uniform" n-test $(\frac{1}{n}1, \ldots, \frac{1}{n}1)$. Alternatively, it may be understood as initialisation like in Example 4.4, given by the state $(\frac{1}{\sqrt{n}}, \ldots, \frac{1}{\sqrt{n}}) \in \mathbb{C}^n$.

Second, the functor $\mathcal{K}\ell_{\mathbb{N}}(\mathcal{D}) \to (\mathbf{CCstar}_{\mathrm{PU}})^{\mathrm{op}}$ preserves block structures, since for $m \in \mathcal{K}\ell_{\mathbb{N}}(\mathcal{D})$ we have:

$$\mathcal{B}_n(\mathbb{C}^m) = (\mathbb{C}^m)^n \cong \mathbb{C}^{n\times m} = \mathbb{C}^{\mathcal{B}_n(m)}. \tag{15}$$

An n-test $p_1, \ldots, p_n \in [0,1]^m$ for $\mathcal{K}\ell_{\mathbb{N}}(\mathcal{D}) \to \mathbf{EMod}^{\mathrm{op}}$ is at the same time an n-test for $(\mathbf{CCstar}_{\mathrm{PU}})^{\mathrm{op}} \to \mathbf{EMod}^{\mathrm{op}}$, since the effects $[0,1]_{\mathbb{C}^m} = [0,1]^m$ of $\mathbb{C}^m \in (\mathbf{CCstar}_{\mathrm{PU}})^{\mathrm{op}}$ are the same as the effects on $m \in \mathcal{K}\ell_{\mathbb{N}}(\mathcal{D})$, so that the diagram on the left commutes.

$$\begin{tikzcd}
& \mathbf{EMod}^{\mathrm{op}} & \\
\mathcal{K}\ell_{\mathbb{N}}(\mathcal{D}) \arrow[ur] \arrow[rr, "\mathbb{C}^{(-)}"'] & & (\mathbf{CCstar}_{\mathrm{PU}})^{\mathrm{op}} \arrow[ul]
\end{tikzcd}
\qquad
\begin{tikzcd}
& \mathbb{C}^m & \\
\mathbb{C}^{\mathcal{B}_n(m)} \arrow[ur, "\mathbb{C}^{char_p}"] \arrow[r, equal, "\sim"] & (\mathbb{C}^m)^n \arrow[r, equal] & \mathcal{B}_n(\mathbb{C}^m) \arrow[ul, "char_p"']
\end{tikzcd}$$

The triangle on the right shows that the characteristic maps are also preserved via (15) in $\mathbf{CCstar}_{\mathrm{PU}}$, where $char_p$ on the left is in $\mathcal{K}\ell_{\mathbb{N}}(\mathcal{D})$, see Example 3.3, and $char_p$ on the right is in the category of C^*-algebras, see Proposition 4.5, using that $\mathbb{C}^m$ is commutative.

5 Matrix block structure on C^*-algebras

For a C^*-algebra A and number $n \in \mathbb{N}$ let $\mathcal{M}_n(A) = A^{n\times n}$ be the vector space of $n \times n$-matrices with entries from A. It is again a C^*-algebra with matrix multiplication, unit and conjugate transpose $(-)^\dagger$. Clearly $\mathcal{M}_1(A) \cong A$, but also $\mathcal{M}_k(\mathcal{M}_n(A)) \cong \mathcal{M}_{k\times n}(A)$. Hence these matrices behave like a block structure.

It turns out that $\mathcal{M}_n$ is not a functor $\mathbf{Cstar}_{\mathrm{PU}} \to \mathbf{Cstar}_{\mathrm{PU}}$, since $\mathcal{M}_n(f)$ need not be positive when f is positive. One therefore calls f *completely positive* when $\mathcal{M}_n(f)$ is positive, for each n. We write $\mathbf{Cstar}_{\mathrm{cPU}} \hookrightarrow \mathbf{Cstar}_{\mathrm{PU}}$ for the (non-full) subcategory of C^*-algebras with completely positive maps between them.

Each MIU-map is completely positive. When $f\colon A \to B$ is a PU-map, where either A or B is commutative, then f is completely positive. One thus requires complete positivity only in the non-commutative PU-case, that is, in a proper quantum setting. The following is the analogue of Proposition 4.3 for matrices.

Proposition 5.1 *Let $\overrightarrow{a_i} = (a_1, \ldots, a_n) \in A^n$ be an n-tuple in a C^*-algebra A. The tuple can be used to form "measurement" and "map" functions.*

(i) *If $\sum_i a_i^* a_i = 1$ there is a completely positive map:*

$$\mathcal{M}_n(A) \xrightarrow{meas(\overrightarrow{a_i})} M \quad \textit{given by} \quad M \longmapsto (a_1^* \ldots a_n^*) M \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix}$$

(ii) *If $a_i^* a_i = 1$ for each i, then there is a completely positive map:*

$$\mathcal{M}_n(A) \xrightarrow{map(\overrightarrow{a_i})} \mathcal{M}_n(A) \quad \textit{given by} \quad M \longmapsto diag(\overrightarrow{a_i^*}) M \, diag(\overrightarrow{a_i})$$

where $\mathrm{diag}(\overrightarrow{a_i})$ *is the diagonal matrix* $\begin{pmatrix} a_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & a_n \end{pmatrix}$. □

We present an example later on in Example 5.5.

Lemma 5.2 *Taking* $n \times n$*-matrices yields a functor* $\mathcal{M}_n \colon \boldsymbol{Cstar}_{cPU} \to \boldsymbol{Cstar}_{cPU}$*, for each* $n > 0$*. It forms a block structure via "in" and "out" natural transformations in a commuting triangle in* $(\boldsymbol{Cstar}_{cPU})^{op}$

$$\begin{tikzcd}
A \arrow[r, "in_n"] \arrow[dr, equal] & \mathcal{M}_n(A) \arrow[d, "out_n"] \\
& A
\end{tikzcd}$$

These natural transformations are given by:

$$in_n(M) = \tfrac{1}{n} tr(M) = \tfrac{1}{n} \textstyle\sum_{i \le n} M_{ii} \qquad out_n(a) = aI_n = \begin{pmatrix} a & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & a \end{pmatrix},$$

where $I_n \in \mathcal{M}_n(A)$ *is the unit/identity matrix. Here,* out_n *is a MIU-map.*

Moreover, the diagonal map $\mathrm{diag} \colon \mathcal{B}_n(A) \to \mathcal{M}_n(A)$ *is a natural transformation that commutes with the* in*'s and* out*'s.*

Proof By definition there is a functor $\mathcal{M}_n \colon \mathbf{Cstar}_{\mathrm{cPU}} \to \mathbf{Cstar}_{\mathrm{PU}}$. We have to prove that $\mathcal{M}_n(f)$ is completely positive, for a completely positive map f. Hence for each k, the map $\mathcal{M}_k(\mathcal{M}_n(f))$ must be positive. But the latter can also be described as $\mathcal{M}_{k \times n}(f)$, via the isomorphism $\mathcal{M}_k \circ \mathcal{M}_n \cong \mathcal{M}_{k \times n}$, which is positive because f is completely positive.

It is a basic fact that the trace map tr is completely positive. Hence so is $in_n = \frac{1}{n} tr$. The map $out_n \colon A \to \mathcal{M}_n(A)$ preserves multiplication and is thus completely positive. Clearly,

$$\big(out \circ^{\mathrm{op}} in\big)(a) = in\big(out(a)\big) = in \begin{pmatrix} a & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & a \end{pmatrix} = \tfrac{1}{n} tr \begin{pmatrix} a & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & a \end{pmatrix} = \tfrac{1}{n} na = a.$$

It is easy to see that $diag$ is natural, *i.e.* that the equation $\mathcal{M}_n(f) \circ diag = diag \circ \mathcal{B}_n(f)$ holds. Moreover, $diag$ commutes with the $\mathcal{B}$ and $\mathcal{M}$ maps:

$$\begin{aligned}
\big(diag \circ^{\mathrm{op}} in^{\mathcal{M}}\big)(\overrightarrow{a_i}) &= in^{\mathcal{M}} \begin{pmatrix} a_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & a_n \end{pmatrix} = \tfrac{1}{n} \textstyle\sum_i a_i = in^{\mathcal{B}}(\overrightarrow{a_i}) \\
\big(out^{\mathcal{B}} \circ^{\mathrm{op}} diag\big)(a) &= diag(a, \ldots, a) = \begin{pmatrix} a & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & a \end{pmatrix} = a \cdot I_n = out^{\mathcal{M}}(a).
\end{aligned}$$

□

The next step is to show that matrices form a logical block structure.

Proposition 5.3 *The matrix block structure* $\mathcal{M}_n$ *on* $(\boldsymbol{Cstar}_{cPU})^{op}$ *is logical, with:*

(i) *the universal* n*-test consisting of positive matrices* $\Omega_i = |\, i \,\rangle\langle\, i \,| \in \mathcal{M}_n(A)$*, clearly with* $\ovee_i \Omega_i = I_n$*;*

(ii) *for an arbitrary n-test $e_i \in [0,1]_A$ a characteristic map* $char_e \colon A \to \mathcal{M}_n(A)$ *in* $(\boldsymbol{Cstar}_{cPU})^{op}$ *given by:*

$$char_e(M) = (\sqrt{e_1} \dots \sqrt{e_n}) M \begin{pmatrix} \sqrt{e_1} \\ \vdots \\ \sqrt{e_n} \end{pmatrix}$$

The characteristic maps for the copower and matrix block structures $\mathcal{B}_n$ and $\mathcal{M}_n$ are related via the diagonal: $\mathrm{diag} \circ^{op} char_e^{\mathcal{M}} = char_e^{\mathcal{B}}$.

Proof Clearly $char_e^{-1}(\Omega_i) = char_e(|\,i\,\rangle\langle\,i\,|) = \sqrt{e_i}\sqrt{e_i} = e_i$. And:

$$\begin{aligned}
\big(\mathrm{diag} \circ^{\mathrm{op}} char_e^{\mathcal{M}}\big)(a_1, \dots, a_n) &= char_e^{\mathcal{M}} \begin{pmatrix} a_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & a_n \end{pmatrix} \\
&= (\sqrt{e_1} \dots \sqrt{e_n}) \begin{pmatrix} a_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & a_n \end{pmatrix} \begin{pmatrix} \sqrt{e_1} \\ \vdots \\ \sqrt{e_n} \end{pmatrix} \\
&= \sqrt{e_1} a_1 \sqrt{e_1} + \cdots + \sqrt{e_n} a_n \sqrt{e_n} \\
&= char_e^{\mathcal{B}}(a_1, \dots, a_n).
\end{aligned}$$

□

The following result collects some standard facts.

Proposition 5.4 *Let $\mathcal{L}(H)$ be the set of bounded linear maps $H \to H$, where H is a Hilbert space. The mapping $H \mapsto \mathcal{L}(H)$ forms a functor*

$$\boldsymbol{Hilb}_{isom} \xrightarrow{\ \mathcal{L}\ } (\boldsymbol{Cstar}_{cPU})^{op}, \tag{16}$$

where $\boldsymbol{Hilb}_{isom}$ is the category of Hilbert spaces with isometries (dagger monos) between them. Each such a dagger mono $f \colon H \rightarrowtail K$ gives a completely positive map $\mathcal{L}(f) = f^\dagger \circ (-) \circ f \colon \mathcal{L}(K) \to \mathcal{L}(H)$. In this situation we have:

$$\mathcal{M}_n(\mathcal{L}(H)) \cong \mathcal{L}(H^n) \cong \mathcal{L}(H \otimes \mathbb{C}^n) \qquad \text{where} \qquad H^n = H \oplus \cdots \oplus H.$$

*Thus $\mathcal{L}$ maps the copower block structure $n \cdot (-)$ on **Hilb**, from Example 2.5, to the matrix block structure $\mathcal{M}_n$ on $(\boldsymbol{Cstar}_{PU})^{op}$. This functor $\mathcal{L}$ also preserves effects and characteristic maps.*

Proof A matrix in $\mathcal{M}_n(\mathcal{L}(H))$ consists of $n \times n$ bounded maps $H \to H$. Since direct sum $\oplus$ is a biproduct for Hilbert spaces, these maps correspond to a single map $H^n \to H^n$, *i.e.* an element of $\mathcal{L}(H^n)$. Next we use that $\mathbb{C}$ is the tensor unit in **Hilb** and that $\otimes$ distributes over $\oplus$ in:

$$\begin{aligned}
H^n \;=\; H \oplus \cdots \oplus H &\cong (H \otimes \mathbb{C}) \oplus \cdots \oplus (H \otimes \mathbb{C}) \\
&\cong H \otimes (\mathbb{C} \oplus \cdots \oplus \mathbb{C}) \;=\; H \otimes \mathbb{C}^n.
\end{aligned}$$

For an isometry (dagger mono) $f \colon H \rightarrowtail K$ in **Hilb** we have $\mathcal{L}(f) = f^\dagger \circ (-) \circ f \colon \mathcal{L}(K) \to \mathcal{L}(H)$. We use $\mathcal{M}_n(B(\mathcal{H})) \cong B(H \otimes \mathbb{C}^n)$, with the map corresponding to $\mathcal{M}_n(\mathcal{L}(f))$ being:

$$\mathcal{L}(K \otimes \mathbb{C}^n) \xrightarrow{(f^\dagger \otimes \mathrm{id}) \circ (-) \circ (f \otimes \mathrm{id})} \mathcal{L}(H \otimes \mathbb{C}^n).$$

We show that if $g \in \mathcal{L}(K \otimes \mathbb{C}^n)$ is positive, then so is $(f^\dagger \otimes \mathrm{id})g(f \otimes \mathrm{id}) \in \mathcal{L}(H \otimes \mathbb{C}^n)$. For a vector $v \in H \otimes \mathbb{C}^n$, write $w = (f \otimes \mathrm{id})v$; then, using that g is positive:

$$\langle\, v\,|(f^\dagger \otimes \mathrm{id})g(f \otimes \mathrm{id})|\,v\,\rangle \;=\; \langle\, (f \otimes \mathrm{id})v\,|g|\,(f \otimes \mathrm{id})v\,\rangle \;=\; \langle\, w\,|g|\,w\,\rangle \;\geq\; 0.$$

The effects associated with the C^*-algebra $\mathcal{L}(H)$ are the effects $\mathcal{E}f(H) = [0,1]_{\mathcal{L}(H)}$ described Example 3.5. Thus the triangle on the left below commutes.

$$\begin{tikzcd}
& \mathbf{EMod}^{\mathrm{op}} & \\
\mathbf{Hilb}_{\mathrm{isom}} \arrow[ur] \arrow[rr, "\mathcal{L}"'] & & (\mathbf{Cstar}_{\mathrm{PU}})^{\mathrm{op}} \arrow[ul]
\end{tikzcd}
\qquad
\begin{tikzcd}
& \mathcal{L}(H) & \\
\mathcal{L}(\mathcal{B}_n(H)) \arrow[ur, "\mathcal{L}(char_A)"] \arrow[rr, equal, "\sim"] & & \mathcal{M}_n(H) \arrow[ul, "char_A"']
\end{tikzcd}$$

For an n-test $A = (A_1, \ldots, A_n)$ in $\mathcal{E}f(H)$, the triangle on the right also commutes in $(\mathbf{Cstar}_{\mathrm{PU}})$. The *char* map on the left is as in Example 3.5, and the one on the right as in Proposition 5.3. As described above, a map $f \colon \mathcal{B}_n(H) \to \mathcal{B}_n(H)$ corresponds to a matrix M_f. Then:

$$\begin{aligned}
char_A^{\mathcal{M}}(M_f) \;&=\; (\sqrt{A_1} \ldots \sqrt{A_n}) \begin{pmatrix} \pi_1 \circ f \circ \kappa_1 & \cdots & \pi_1 \circ f \circ \kappa_n \\ \vdots & & \vdots \\ \pi_n \circ f \circ \kappa_1 & \cdots & \pi_n \circ f \circ \kappa_n \end{pmatrix} \begin{pmatrix} \sqrt{A_1} \\ \vdots \\ \sqrt{A_n} \end{pmatrix} \\
&\overset{(*)}{=}\; [\sqrt{A_1} \ldots \sqrt{A_n}] \circ f \circ \langle \sqrt{A_1} \ldots \sqrt{A_n} \rangle \\
&=\; \langle \sqrt{A_1} \ldots \sqrt{A_n} \rangle^\dagger \circ f \circ \langle \sqrt{A_1} \ldots \sqrt{A_n} \rangle \\
&=\; \mathcal{L}(\langle \sqrt{A_1} \ldots \sqrt{A_n} \rangle)(f) \\
&=\; \mathcal{L}(char_A^{\mathcal{B}})(f).
\end{aligned}$$

The marked equation $\overset{(*)}{=}$ involves some elementary calculations with biproducts $\oplus$ in **Hilb**. □

Example 5.5 Consider the "identity" and "negation" matrices $I_2 = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ and $X = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$ as elements $I_2, X \in \mathcal{L}(\mathbb{C}^2)$. The "map" operation from Proposition 5.1 (ii) yields:

$$\mathcal{M}_2(\mathcal{L}(\mathbb{C}^2)) \xrightarrow{map(I_2,X)} \mathcal{M}_2(\mathcal{L}(\mathbb{C}^2)) \quad \text{given by} \quad M \longmapsto \left(\begin{smallmatrix} I_2 & 0 \\ 0 & X \end{smallmatrix}\right) M \left(\begin{smallmatrix} I_2 & 0 \\ 0 & X \end{smallmatrix}\right)$$

Via the isomorphism $\mathcal{M}_2(\mathcal{L}(\mathbb{C}^2)) \cong \mathcal{L}(\mathbb{C}^4)$ this is the operation $\mathcal{L}(\mathsf{CNOT}) \colon \mathcal{L}(\mathbb{C}^4) \to \mathcal{L}(\mathbb{C}^4)$, where CNOT is the "conditional negation" matrix:

$$\mathsf{CNOT} = \left(\begin{smallmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{smallmatrix}\right) = \left(\begin{smallmatrix} I_2 & 0 \\ 0 & X \end{smallmatrix}\right).$$

Remark 5.6 The category $\mathbf{Cstar}_{\mathrm{cPU}}$ also has monoidal structure. In fact, there is a "minimal" and a "maximal" tensor $A \otimes B$, but as long as either A or B is finite-dimensional, they coincide (with $\mathbb{C}$ as tensor unit). Via these tensors we can see a closer analogy between the copower and matrix block structures $\mathcal{B}$ and $\mathcal{M}$ on C^*-algebras, namely:

$$\mathcal{B}_n(A) = A^n \cong \mathbb{C}^n \otimes A = \mathcal{B}_n(\mathbb{C}) \otimes A \qquad \text{and} \qquad \mathcal{M}_n(A) \cong \mathcal{M}_n(\mathbb{C}) \otimes A.$$

In particular, for a Hilbert space H tensors are preserved:

$$\mathcal{L}(\mathbb{C}^n \otimes H) \cong \mathcal{M}_n(H) \cong \mathcal{M}_n(\mathbb{C}) \otimes \mathcal{L}(H) = \mathcal{L}(\mathbb{C}^n) \otimes \mathcal{L}(H).$$

6 Examples and discussion

So far we have seen examples of block structures in a non-deterministic and probabilistic setting — in the Kleisli categories $\mathcal{K}\ell(\mathcal{P})$ and $\mathcal{K}\ell(\mathcal{D})$ — and also in a quantum setting, in the categories of Hilbert spaces and of C^*-algebras. In the latter setting we have seen two block structures, namely copower $\mathcal{B}_n(A) = n \cdot A$ and matrix $\mathcal{M}_n(A)$. It seems that $\mathcal{B}_n$ is most appropriate in a commutative/probabilistic setting, and $\mathcal{M}_n$ in a quantum setting, because:

- $\mathcal{B}_n$ is a comonad, involving a copying operation; $\mathcal{M}_n$ is not a comonad, since copying is impossible in a non-commutative setting, see *e.g.* [17].
- The functor $\mathcal{K}\ell(\mathcal{D}) \to (\mathbf{CCstar}_{\mathrm{PU}})^{\mathrm{op}}$ putting probabilistic transitions in a C^*-algebraic context commutes with $\mathcal{B}_n$.
- The functor $\mathbf{Hilb}_{\mathrm{isom}} \to (\mathbf{Cstar}_{\mathrm{cPU}})^{\mathrm{op}}$ from (16) commutes with $\mathcal{M}_n$.

The issue of which block structure to use, in which situation, remains unclear and will be further explored in follow-up research. In the remainder of this section we briefly investigate how block structures can be used to describe familiar quantum protocols like superdense coding and teleportation as maps in the category of C^*-algebras. Such descriptions can be used to represent the protocols in computer algebra tools, for simulation and verification. The whole point that we are trying to suggest is that logical blocks may form a clean language construct in a future (quantum) programming language.

We start by recalling some basic material. The Bell basis of $\mathbb{C}^4$ is given by the vectors:

$$\begin{aligned} |\, b_1 \rangle &= \tfrac{1}{\sqrt{2}}(|\, 00 \rangle + |\, 11 \rangle) \qquad & |\, b_2 \rangle &= \tfrac{1}{\sqrt{2}}(|\, 01 \rangle + |\, 10 \rangle) \\ |\, b_3 \rangle &= \tfrac{1}{\sqrt{2}}(|\, 00 \rangle - |\, 11 \rangle) \qquad & |\, b_4 \rangle &= \tfrac{1}{\sqrt{2}}(|\, 01 \rangle - |\, 10 \rangle) \end{aligned}$$

The associated projections $e_i = |\, b_i \rangle\langle\, b_i \,| \in \mathcal{E}f(\mathbb{C}^4)$ can be described by the matrices:

$$e_1 = \tfrac{1}{2}\begin{pmatrix} 1&0&0&1\\ 0&0&0&0\\ 0&0&0&0\\ 1&0&0&1 \end{pmatrix} \quad e_2 = \tfrac{1}{2}\begin{pmatrix} 0&0&0&0\\ 0&1&1&0\\ 0&1&1&0\\ 0&0&0&0 \end{pmatrix} \quad e_3 = \tfrac{1}{2}\begin{pmatrix} 1&0&0&-1\\ 0&0&0&0\\ 0&0&0&0\\ -1&0&0&1 \end{pmatrix} \quad e_4 = \tfrac{1}{2}\begin{pmatrix} 0&0&0&0\\ 0&1&-1&0\\ 0&-1&1&0\\ 0&0&0&0 \end{pmatrix}$$

They satisfy $e_1 \ovee e_2 \ovee e_3 \ovee e_4 = \mathrm{id}$ and thus form a 4-test in $\mathcal{E}f(\mathbb{C}^4) = [0,1]_{\mathcal{L}(\mathbb{C}^4)}$. Since $e_i^2 = e_i$ we have $\sqrt{e_i} = e_i$. Further, because the Bell basis is orthogonal, we have $e_i e_j = 0$ for $i \neq j$. We shall write $\mathit{char}_{\mathit{Bell}}$ in $(\mathbf{Cstar}_{\mathrm{cPU}})^{\mathrm{op}}$ for the associated measurement operation $\mathcal{L}(\mathbb{C}^4) \to \mathcal{M}_4(\mathcal{L}(\mathbb{C}^4))$.

Next we need the four Pauli matrices in $\mathcal{L}(\mathbb{C}^2)$:

$$\sigma_1 = \left(\begin{smallmatrix} 1&0\\ 0&1 \end{smallmatrix}\right) \qquad \sigma_2 = X = \left(\begin{smallmatrix} 0&1\\ 1&0 \end{smallmatrix}\right) \qquad \sigma_3 = Z = \left(\begin{smallmatrix} 1&0\\ 0&-1 \end{smallmatrix}\right) \qquad \sigma_4 = XZ = \left(\begin{smallmatrix} 0&-1\\ 1&0 \end{smallmatrix}\right).$$

They all satisfy $\sigma_i^\dagger \sigma_i = I_2$, and may thus be used in "map" constructions, like in Propositions 4.3 and 5.1

6.1 Superdense coding

What the superdense coding algorithm of [2] achieves is sending two classical bits via one (entangled) qubit. Two parties, Alice and Bob each possess one qubit of a shared entangled (Bell) state. Alice applies one of 4 operations σ_i to her qubit — thus encoding one the four options $i \in 4$ given by 2 classical bits — and sends the result to Bob. Through the local operations, represented as $\sigma_i \otimes \mathrm{id} \in \mathcal{L}(\mathbb{C}^4)$, Alice affects the shared state. By performing a Bell measurement Bob can find out which of the four operations σ_i was applied by Alice, and thus which $i \in 4$ is transmitted.

Our block-based representation of the superdense coding protocol consists of the following four maps in the category $(\mathbf{Cstar}_{\mathrm{PU}})^{\mathrm{op}}$.

$$\begin{tikzcd}
\mathcal{L}(\mathbb{C}^4) \arrow[d, "in_4^{\mathcal{B}}"'] & & \mathcal{B}_4(\mathcal{L}(\mathbb{C}^4)) \\
\mathcal{B}_4(\mathcal{L}(\mathbb{C}^4)) \arrow[r, "map(\overrightarrow{\sigma_i \otimes \mathrm{id}})"] & \mathcal{B}_4(\mathcal{L}(\mathbb{C}^4)) \arrow[r, "\mathcal{B}_4(char_{Bell})"] & \mathcal{B}_4(\mathcal{M}_4(\mathcal{L}(\mathbb{C}^4))) \arrow[u, "\mathcal{B}_4(out_4^{\mathcal{M}})"']
\end{tikzcd} \tag{17}$$

First a copower 4-block is opened to deal with the four classical options (corresponding to the two classical bits at hand). In each of these four options Alice performs one of the operations σ_i, only to her part of the shared state, via $\sigma_i \otimes \mathrm{id}$. These operations are combined in a single one via "map". At this stage Alice transfers her qubit to Bob, and Bob owns the whole state. In each of the four block options he performs a Bell measurement. Then he closes the outer block. The outcome of these Bell measurements distinguishes the various block options and enables Bob to recognise these options.

The computation (17) in $(\mathbf{Cstar}_{\mathrm{PU}})^{\mathrm{op}}$ consists of a computation $\mathcal{B}_4(\mathcal{L}(\mathbb{C}^4)) \to \mathcal{L}(\mathbb{C}^4)$ that computes the weakest precondition. We shall compute it with the above Bell projections (e_1, e_2, e_3, e_4) as input to this computation going backwards:

$$\begin{aligned}
&\big(\mathcal{B}_4(out_4^{\mathcal{M}}) \circ^{\mathrm{op}} \mathcal{B}_4(char_{Bell}) \circ^{\mathrm{op}} map(\overrightarrow{\sigma_i \otimes \mathrm{id}}) \circ^{\mathrm{op}} in_4^{\mathcal{B}}\big)(e_1, e_2, e_3, e_4) \\
&\quad = \big(in_4^{\mathcal{B}} \circ map(\overrightarrow{\sigma_i \otimes \mathrm{id}}) \circ (char_{Bell})^4\big)\big(e_1 I_4, e_2 I_4, e_3 I_4, e_4 I_4\big) \\
&\quad = \big(in_4^{\mathcal{B}} \circ map(\overrightarrow{\sigma_i \otimes \mathrm{id}})\big)\big(\textstyle\sum_i \sqrt{e_i} e_1 \sqrt{e_i}, \sum_i \sqrt{e_i} e_2 \sqrt{e_i}, \sum_i \sqrt{e_i} e_3 \sqrt{e_i}, \sum_i \sqrt{e_i} e_4 \sqrt{e_i}\big) \\
&\quad = \big(in_4^{\mathcal{B}} \circ map(\overrightarrow{\sigma_i \otimes \mathrm{id}})\big)(e_1, e_2, e_3, e_4) \\
&\quad = in_4^{\mathcal{B}}\big((\sigma_1 \otimes \mathrm{id})^\dagger e_1 (\sigma_1 \otimes \mathrm{id}), (\sigma_2 \otimes \mathrm{id})^\dagger e_2 (\sigma_2 \otimes \mathrm{id}), \\
&\qquad\qquad (\sigma_3 \otimes \mathrm{id})^\dagger e_3 (\sigma_3 \otimes \mathrm{id})(\sigma_4 \otimes \mathrm{id})^\dagger e_4 (\sigma_4 \otimes \mathrm{id})\big) \\
&\quad \overset{(*)}{=} in_4^{\mathcal{B}}(e_1, e_1, e_1, e_1) \\
&\quad = e_1.
\end{aligned}$$

The equalities $(\sigma_i^\dagger \otimes \mathrm{id}) e_i (\sigma_i \otimes \mathrm{id}) = e_1$ used in marked equation $\overset{(*)}{=}$ are left to the reader.

This calculation for (17) can be interpreted as follows. In order to get as postcondition (e_1, e_2, e_3, e_4), one has to start the computation with precondition e_1. This precondition $e_1 = |\, b_1 \,\rangle\langle\, b_1 \,|$ for $|\, b_1 \,\rangle = \frac{1}{\sqrt{2}}(|\,00\,\rangle + |\,11\,\rangle)$ is the shared Bell state that usually serves as starting point for super dense coding.

6.2 Teleportation

For the teleportation protocol (see *e.g.* [18]) we open a "matrix" block via initialisation. The bell basis vector $|\,b_1\,\rangle = \frac{1}{\sqrt{2}}(|\,00\,\rangle + |\,11\,\rangle) \in \mathbb{C}^4$ gives rise to a (dagger monic) map $\mathrm{id} \otimes |\,b_1\,\rangle \colon \mathbb{C}^2 \rightarrowtail \mathbb{C}^2 \otimes \mathbb{C}^4$. By applying the functor $\mathcal{L}$ from Proposition 5.4 we obtain:

$$\mathcal{L}(\mathbb{C}^2) \longrightarrow \mathcal{L}(\mathbb{C}^2 \otimes \mathbb{C}^4) \cong \mathcal{M}_4(\mathcal{L}(\mathbb{C}^2))$$

This is the first map in the protocol below.

$$\begin{tikzcd}
\mathcal{L}(\mathbb{C}^2) \arrow[d] & & & \mathcal{L}(\mathbb{C}^2) \\
\mathcal{M}_4(\mathcal{L}(\mathbb{C}^2)) \arrow[r, "\mathit{char}_{\mathit{Bell}}"] & \mathcal{B}_4(\mathcal{M}_4(\mathcal{L}(\mathbb{C}^2))) \arrow[r, "\mathcal{B}_4(\mathit{out}_4^{\mathcal{M}})"] & \mathcal{B}_4(\mathcal{L}(\mathbb{C}^2)) \arrow[r, "\mathit{map}(\overrightarrow{\sigma_i})"] & \mathcal{B}_4(\mathcal{L}(\mathbb{C}^2)) \arrow[u, "\mathit{out}_4^{\mathcal{B}}"']
\end{tikzcd}$$

In this case, after initialisation Alice does a measurement $\mathit{char}_{\mathit{Bell}}$ giving a copower block $\mathcal{B}_4$ in order to transfer two bits of information to Bob. Here we consider the above matrices e_i as matrices over $\mathcal{L}(\mathbb{C}^2)$. In each of the resulting 4 block options Bob does an adjustment, with the Pauli matrices σ_i. It can be shown that the resulting map $\mathcal{L}(\mathbb{C}^2) \to \mathcal{L}(\mathbb{C}^2)$ is the identity.

Conclusions

This paper presents the first steps towards understanding the structure and role of blocks and predicates in non-deterministic / probabilistic / quantum programming. The opening of blocks via characteristic maps (measurements) induced by n-tests in effect algebras is common in these approaches. For the particular case of "von Neumann" n-tests of projections this can be described via Eilenberg-Moore coalgebras. In the general case there is much variation that requires further investigation.

Acknowledgements

Thanks to Robert Furber and Jorik Mandemaker for helpful discussions.

References

[1] T. Altenkirch and J. Grattage. A functional quantum programming language. In *Logic in Computer Science*, pages 249–259. IEEE, Computer Science Press, 2005.

[2] C. Bennett and S.J. Wiesner. Communication via one- and two-particle operators on Einstein-Podolsky-Rosen states. *Phys. Review Letters*, 69(20):2881–2884, 1992.

[3] B. Coecke and D. Pavlović. Quantum measurements without sums. In G. Chen, L. Kauffman, and S. Lamonaco, editors, *Mathematics of Quantum Computing and Technology*, pages 559–596. Taylor and Francis, 2008.

[4] A. Dvurečenskij and S. Pulmannová. *New Trends in Quantum Structures*. Kluwer Acad. Publ., Dordrecht, 2000.

[5] D. J. Foulis and M.K. Bennett. Effect algebras and unsharp quantum logics. *Found. Physics*, 24(10):1331–1352, 1994.

[6] R. Furber and B. Jacobs. From Kleisli categories to commutative C^*-algebras: Probabilistic Gelfand duality. See arxiv.org/abs/1303.1115, 2013.

[7] S. Gay. Quantum programming languages: survey and bibliography. *Math. Struct. in Comp. Sci.*, 16:581–600, 2006.

[8] M. Giry. A categorical approach to probability theory. In B. Banaschewski, editor, *Categorical Aspects of Topology and Analysis*, number 915 in Lect. Notes Math., pages 68–85. Springer, Berlin, 1982.

[9] T. Heinosaari and M. Ziman. *The Mathematical Language of Quantum Theory. From Uncertainty to Entanglement.* Cambridge Univ. Press, 2012.

[10] B. Jacobs. New directions in categorical logic, for classical, probabilistic and quantum logic. See arxiv.org/abs/1205.3940, 2012.

[11] B. Jacobs. Measurable spaces and their effect logic. In *Logic in Computer Science*. IEEE, Computer Science Press, 2013.

[12] B. Jacobs and J. Mandemaker. Coreflections in algebraic quantum logic. *Found. of Physics*, 42(7):932–958, 2012.

[13] B. Jacobs and J. Mandemaker. Relating operator spaces via adjunctions. In J. Chubb Reimann, V. Harizanov, and A. Eskandarian, editors, *Logic and Algebraic Structures in Quantum Computing and Information*, Lect. Notes in Logic. Cambridge Univ. Press, 2013. See arxiv.org/abs/1201.1272.

[14] P. Johnstone. *Stone Spaces.* Number 3 in Cambridge Studies in Advanced Mathematics. Cambridge Univ. Press, 1982.

[15] K. Kraus. *States, Effects, and Operations.* Springer Verlag, Berlin, 1983.

[16] G. Ludwig. *Foundations of Quantum Mechanics I.* Springer Verlag, New York, 1983.

[17] H. Maassen. Quantum probability and quantum information theory. In F. Benatti, M. Fannes, R. Floreanini, and D. Petritis, editors, *Quantum Information, Computation and Cryptography*, number 808 in Lect. Notes Physics, pages 65–108. Springer, Berlin, 2010.

[18] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information.* Cambridge Univ. Press, 2000.

[19] P. Selinger. Towards a quantum programming language. *Math. Struct. in Comp. Sci.*, 14(4):527–586, 2004.

[20] P. Selinger and B. Valiron. A lambda calculus for quantum computation with classical control. *Math. Struct. in Comp. Sci.*, 16(3):527–552, 2006.

[21] P. Selinger and B. Valiron. Quantum lambda calculus. In S. Gay and I. Mackie, editors, *Semantical Techniques in Quantum Computation*, pages 135–172. Cambridge Univ. Press, 2010.

[22] J. Tomiyama. On the projection of norm one in W^*-algebras. *Proc. Japan Acad.*, 10:608–612, 1957.

Rational Operational Models

Stefan Milius[1]

Lehrstuhl für Theoretische Informatik, Friedrich-Alexander Universität Erlangen-Nürnberg, Germany

Marcello M. Bonsangue[2]

LIACS – Leiden University and Centrum Wiskunde en Informatica (CWI), The Netherlands

Robert S.R. Myers[3]

Institut für Theoretische Informatik, TU Braunschweig, Germany

Jurriaan Rot[4]

LIACS – Leiden University and Centrum Wiskunde en Informatica (CWI), The Netherlands

Abstract

GSOS is a specification format for well-behaved operations on transition systems. Aceto introduced a restriction of this format, called *simple GSOS*, which guarantees that the associated transition system is locally finite, i.e. every state has only finitely many different descendent states (i.e. states reachable by a sequence of transitions).

The theory of *coalgebras* provides a framework for the uniform study of systems, including labelled transition systems but also, e.g. weighted transition systems and (non-)deterministic automata. In this context GSOS can be studied at the general level of distributive laws of syntax over behaviour. In the present paper we generalize Aceto's result to the setting of coalgebras by restricting abstract GSOS to *bipointed specifications*. We show that the operational model of a bipointed specification is locally finite, even for specifications with infinitely many operations which have finite dependency. As an example, we derive a concrete format for operations on regular languages and obtain for free that regular expressions have finitely many derivatives modulo the equations of join semilattices.

Keywords: coalgebra, distributive law, regular process, simple GSOS, rational behaviour

1 Introduction

GSOS [13] is a popular specification format for operations on transition systems, which guarantees that bisimulation is a congruence. Every GSOS specification induces an *operational model*, which is a concrete transition system on the closed

[1] Email: mail@stefan-milius.eu

[2] Email: marcello@liacs.nl

[3] Email: me.robmyers@gmail.com

[4] Email: jrot@liacs.nl. The research of this author has been funded by the Netherlands Organisation for Scientific Research (NWO), CoRE project, dossier number: 612.063.920.

terms of the syntax. Aceto's simple GSOS [1] is a restriction of this format which guarantees the operational model to be locally finite. This means that any state in this model is contained in a finite subsystem, i.e. it has only finitely many different descendent states. Consequently, the behaviour of each term is some kind of regular tree modulo bisimilarity. Simple GSOS rules differ from ordinary GSOS in that the target of a conclusion is either a single operation or a variable, rather than an arbitrary term. Moreover, while the number of operations can be infinite, each operation may only depend on finitely many others. Most operations used in practice can be specified in simple GSOS [2].

Operations that preserve finiteness are of considerable importance in automata theory. In order to provide a uniform mathematical treatment of operations on different types of systems, including those from automata theory, we use the theory of universal coalgebra, where the type of a system is completely specified by an endofunctor F. In this context, the *rational fixpoint* of a endofunctor F on Set is that subcoalgebra of the final F-coalgebra which consists of the behaviours of all finite F-coalgebras. Bipointed specifications were introduced in [14] as a format which, for a given finite signature of finite arity operations, defines algebraic operations on the rational fixpoint. This provides an easy syntactic criterion for the preservation of finite behaviour, whose format is a restriction of Turi's and Plotkin's generalization of GSOS via distributive laws [31,20]. Under the assumption that the signature is finite, bipointed specifications for labelled transition systems coincide with simple GSOS. However, the operational model was not considered in [14].

In this paper we complete the generalization of Aceto's results: (a) we extend the results of [14] from specifications for finitely many algebraic operations to specifications that may define infinitely many operations, but with *finite dependency* (cf. [2]); (b) we prove that for a bipointed specification having finite dependency its operational model is locally finite. Result (a) allows, e.g. to treat *all* real numbers as constants in the stream calculus [28], while (b) gives a construction of a finite model for each term, thus paving the way for decidability results.

For the set functor whose coalgebras are deterministic automata, the rational fixpoint is carried by the set of regular languages. At this point one might expect that all the operators of regular expressions might be specified by bipointed specifications for this functor. However, the corresponding rule format is not expressive enough to capture concatenation or the Kleene star. So as a final result we derive a concrete rule format for operations on regular languages, by instantiating our results in the category of join semilattices. Operations defined by rules in this format preserve regular languages, examples being the shuffle product or sequential composition. In fact the format allows us to define the behaviour of regular expressions. Consequently we obtain for free the well-known result [16] that regular expressions modulo the axioms of join semilattices have only finitely many derivatives.

Due to space constraints we omit all proofs; a version of this paper containing all proofs is available at http://www.stefan-milius.eu.

2 Preliminaries

We assume that the reader is familiar with basic notions from category theory, including (initial) algebras and (final) coalgebras for endofunctors. Let us now fix notation and briefly mention some examples. We denote by Set the category of sets and functions and by Jsl the category of join semilattices and their morphisms.

We denote the initial algebra for a functor $\Sigma : \mathcal{A} \to \mathcal{A}$ by $\iota : \Sigma(\mu\Sigma) \to \Sigma$. In most cases in this paper, Σ will be a polynomial functor on Set given by a (finitary, yet not necessarily *finite*) signature of operation symbols, each with prescribed finite arity. Algebras and homomorphisms for such a functor are precisely the general algebras and homomorphisms for the signature.

The final coalgebra for a functor $F : \mathcal{A} \to \mathcal{A}$ is denoted by $t : \nu F \to F(\nu F)$. We consider several examples of coalgebras for $\mathcal{A} = \mathsf{Set}$ (see [27] for many more):

Example 2.1 (1) Deterministic automata with input alphabet A are coalgebras for $FX = 2 \times X^A$, where $2 = \{0, 1\}$. The final coalgebra is carried by the set of languages $\mathcal{P}(A^*)$.

(2) Finitely branching labelled transition systems (LTS) with actions from the set A are coalgebras for $FX = \mathcal{P}_{\mathsf{f}}(A \times X)$. The final coalgebra for F exists and can be thought of as consisting of processes modulo strong bisimilarity of Milner [25].

(3) Weighted transition systems (WTS) are labelled transition systems where transitions have weights (modelling multiplicities, costs, probabilities, etc.) in a monoid $\mathbb{M} = \langle M, +, 0\rangle$. They can be seen as coalgebras (see e.g. Klin [19]): one considers the functor $\mathcal{F}_{\mathbb{M}}$, which acts on a set X and a function $f : X \to Y$ as $\mathcal{F}_{\mathbb{M}}(X) = \{\phi : X \to M \mid \phi \text{ has finite support}\}$ and $\mathcal{F}_{\mathbb{M}} f(\phi)(y) = \sum_{x \in f^{-1}(y)} \phi(x)$. Weighted transition systems with actions from the set A are then precisely coalgebras for $FX = (\mathcal{F}_{\mathbb{M}} X)^A$.

2.1 Locally finitely presentable coalgebras. We are interested in algebraic operations on regular behaviour, i.e. behaviour of *finite* coalgebras (S, f) for a functor F. As previously in [14] we present our results for endofunctors on general categories $\mathcal{A}$ in which it makes sense to talk about "finite" objects and the ensuing rational behaviour of "finite" coalgebras. So we work with the *locally finitely presentable* categories of Gabriel and Ulmer [17] (see also Adámek and Rosický [7]), and we now briefly recall the basics.

A functor $F : \mathcal{A} \to \mathcal{B}$ is called *finitary* if $\mathcal{A}$ has and F preserves filtered colimits. An object X of a category $\mathcal{A}$ is called *finitely presentable* if its hom-functor $\mathcal{A}(X, -)$ is finitary. A category $\mathcal{A}$ is *locally finitely presentable* (lfp) if (a) it is cocomplete, and (b) it has a set of finitely presentable objects such that every object of $\mathcal{A}$ is a filtered colimit of objects from that set.

Example 2.2 (1) Examples of lfp categories include the category Set, the category of posets and monotone functions, and the category of (multi)graphs and graph morphisms. Their finitely presentable objects are the finite sets, finite posets and finite graphs, respectively.

(2) Fix any finitary signature and also a set of equations between terms over this signature. This induces a finitary variety i.e. a category whose objects are the

algebras for this signature which satisfy the equations, e.g. groups, monoids, join semilattices etc. Its morphisms are the usual algebra morphisms for the signature. Such categories are lfp: the finitely presentable objects are those algebras presented by finitely many generators and finitely many relations.

(3) As a special case consider *locally finite* varieties, where the free algebras on finitely many generators are finite. Examples include join semilattices, distributive lattices, boolean algebras and the two-sorted variety of multigraphs. Here the finitely presentable objects are precisely the finite algebras.

(4) Another special case of point (2) is the category $\mathsf{Vec}_{\mathbb{F}}$ of vector spaces over any fixed field $\mathbb{F}$, where the finitely presentable objects are precisely the finite dimensional vector spaces.

Remark 2.3 On the category Set, a finitary functor is determined by its behaviour on finite sets. More precisely, a functor $F : \mathsf{Set} \to \mathsf{Set}$ is finitary iff it is *bounded* (see, e.g. Adámek and Trnková [10]), i.e. for every set X and every element $t \in FX$, there exists a finite subset $i : Y \hookrightarrow X$ such that $t \in Fi[FY] \subseteq FX$.

Example 2.4 The *finite* powerset functor $\mathcal{P}_{\mathsf{f}}$ is finitary, whereas the ordinary powerset functor $\mathcal{P}$ is not. The functor $FX = X^A$ is finitary if and only if A is a finite set. More generally, the class of finitary endofunctors on Set contains all constant functors and the identity functor, and it is closed under finite products, arbitrary coproducts and composition. Thus, a polynomial functor Σ is finitary iff every operation symbol of the corresponding signature has finite arity (but there may be infinitely many operations). The functor $FX = \mathbb{R} \times X$ is finitary both on Set and on $\mathsf{Vec}_{\mathbb{R}}$.

Assumption 2.5 Throughout the rest of this paper, we assume, unless stated otherwise, that $\mathcal{A}$ is a locally finitely presentable category and $F : \mathcal{A} \to \mathcal{A}$ is a finitary functor. So F has a final coalgebra $t : \nu F \to F(\nu F)$ (see Makkai and Paré [23]).

For a functor F on an lfp category $\mathcal{A}$, the notion of a “finite” coalgebra is captured by requiring the carrier to be finitely presentable. That is, we denote by $\mathsf{Coalg}_{\mathsf{f}}(F)$ the full subcategory of $\mathsf{Coalg}(F)$ consisting of those F-coalgebras $f : S \to FS$ whose carrier S is a finitely presentable object in $\mathcal{A}$. In order to talk about the behaviour of finite coalgebras in this setting, we would like to consider a coalgebra that is final amongst all coalgebras in $\mathsf{Coalg}_{\mathsf{f}}(F)$. However, $\mathsf{Coalg}_{\mathsf{f}}(F)$ need not have a final object; for example, in the case of deterministic automata (see Example 2.1(1)), the desired final coalgebra for finite automata should be formed by all regular languages, but this coalgebra is itself not finite. For this reason we take the closure of $\mathsf{Coalg}_{\mathsf{f}}(F)$ under filtered colimits in $\mathsf{Coalg}(F)$, in which the desired final object exists. It is often useful to view these filtered colimits as directed unions of machines, taken at the level of their carrier. We will write $\mathsf{Coalg}_{\mathsf{lfp}}(F)$ for this closure. The objects of $\mathsf{Coalg}_{\mathsf{lfp}}(F)$ were called *locally finitely presentable* coalgebras in [24,15,14]; they are precisely the filtered colimits of diagrams over $\mathsf{Coalg}_{\mathsf{f}}(F)$, i.e. colimits of filtered diagrams of the form $\mathcal{D} \to \mathsf{Coalg}_{\mathsf{f}}(F) \hookrightarrow \mathsf{Coalg}(F)$.

Example 2.6 We recall from [24,15] concrete descriptions of the objects of $\mathsf{Coalg}_{\mathsf{lfp}}(F)$ in some categories of interest.

(1) A coalgebra for a functor on Set is locally finitely presentable iff it is *locally finite*, i.e. every finite subset of its carrier is contained in a finite subcoalgebra.

(2) For an endofunctor on a locally finite variety, a coalgebra is locally finitely presentable iff every finite subalgebra of its carrier lies in a finite subcoalgebra.

(3) A coalgebra (S, f) for a functor on $\mathsf{Vec}_{\mathbb{F}}$ is locally finitely presentable iff every finite dimensional subspace of its carrier S is contained in a subcoalgebra of (S, f) whose carrier is finite dimensional.

Recall from [23], that the Ind-completion of a category is the free completion of that category under filtered colimits. We will make use of the following non-trivial fact:

Theorem 2.7 *The category* $\mathsf{Coalg}_{\mathsf{lfp}}(F)$ *is the* Ind*-completion of* $\mathsf{Coalg}_{\mathsf{f}}(F)$*.*

2.2 The rational fixpoint. Clearly, the category $\mathsf{Coalg}_{\mathsf{lfp}}(F)$ has a final object given by the filtered colimit of the inclusion functor $\mathsf{Coalg}_{\mathsf{f}}(F) \hookrightarrow \mathsf{Coalg}(F)$. We denote this coalgebra by $r : \rho F \to F(\rho F)$. This coalgebra captures the behaviour of all coalgebras in $\mathsf{Coalg}_{\mathsf{f}}(F)$. It has been shown in [5] that it is a fixpoint of F, i. e., its structure morphism r is an isomorphism. Following [24,15] we call the coalgebra $(\rho F, r)$ the *rational fixpoint* of F.

Remark 2.8 For $\mathcal{A} = \mathsf{Set}$ the rational fixpoint ρF is the union of all images $f^\dagger[S] \subseteq \nu F$, where $f : S \to FS$ ranges over the *finite* F-coalgebras and $f^\dagger : S \to \nu F$ is the unique coalgebra homomorphism (see [5, Proposition 4.6 and Remark 4.3]). So, in particular, we see that ρF is a subcoalgebra of νF.

For endofunctors on different categories than Set, this need not be the case as shown in [15, Example 3.15]. However, for functors preserving monomorphisms on categories of vector spaces over a field and on locally finite varieties such as Jsl the rational fixpoint always is a subcoalgebra of νF (see [15, Proposition 3.12]).

Example 2.9 We give a number of examples of ρF; for more, see [5,15].

(1) For the functor $FX = \mathbb{R} \times X$ on Set whose final coalgebra is carried by the set $\mathbb{R}^\omega$ of all streams over $\mathbb{R}$, the rational fixpoint consists of all streams that are *eventually periodic*, i.e., of the form $\sigma = vwwww\ldots$ for words $v \in \mathbb{R}^*$ and $w \in \mathbb{R}^+$. For the similar functor $FV = \mathbb{R} \times V$ on the category of vector spaces over $\mathbb{R}$, the rational fixpoint consists of all *rational streams* (e. g., Rutten [29]).

(2) The carrier of the rational fixpoint of the deterministic automata functor $FX = 2 \times X^A$ is the set of all languages accepted by *finite* automata, viz. the set of all *regular* languages. If we define F instead on the category Jsl of join semilattices, its rational fixpoint is still given by all regular languages, this time with the join semilattice structure given by union and $\emptyset$.

(3) For $FX = \mathcal{P}_{\mathsf{f}}(A \times X)$ on Set the rational fixpoint contains all *finite-state* processes (modulo bisimilarity); more precisely, ρF is the coproduct of all *finite* F-coalgebras modulo the largest bisimulation.

(4) For the functor $FX = (\mathcal{F}_{\mathbb{M}} X)^A$ of weighted transition systems the rational fixpoint is obtained as the coproduct of all finite WTS's modulo weighted bisimilarity.

2.3 Bipointed specifications. In [14] we introduced *bipointed specifications*, which are natural transformations of the form $\Sigma(F \times Id) \to F(\Sigma + Id)$, where $\Sigma : \mathcal{A} \to \mathcal{A}$ is a given functor. We also showed that for Σ a polynomial endofunctor for a finite signature on Set and for $FX = \mathcal{P}_\mathsf{f}(A \times X)$ bipointed specifications are equivalent to transition system specifications in the simple GSOS format of Aceto [1]. In order to understand Aceto's theorem below and to give a first intuition on bipointed specifications we now recall GSOS and simple GSOS. Given a signature Σ, a GSOS rule for an operator $f \in \Sigma$ of arity n is of the form (1) where m is the number of positive premises, l is the number of negative premises, and

$$\frac{\{x_{i_j} \overset{a_j}{\to} y_j\}_{j=1..m} \qquad \{x_{i_k} \overset{b_k}{\not\to}\}_{k=1..l}}{f(x_1, \dots, x_n) \overset{c}{\to} t} \tag{1}$$

$a_1, \dots, a_m, b_1, \dots, b_l, c \in A$ are labels. The variables $x_1, \dots, x_n, y_1, \dots, y_m$ are pairwise distinct; let V denote the set of these variables. Finally t is a Σ-term over variables in V. In the *simple GSOS* format, t is restricted to be either a variable in V or a flat term $g(z_1, \dots, z_p)$, where g is a p-ary operation symbol in Σ and $z_1, \dots, z_p \in V$. Additionally there is a finiteness condition on the dependency of operators, which we recall below in Section 4. Examples of GSOS rules which adhere to the simple GSOS format include the parallel operator, choice, action prefixing, relabelling and many more.

In the mathematical operational semantics of Turi and Plotkin [31] (see also Bartels [12]) one considers for a specification in the form of a natural transformation as above (and more general formats; see Klin [20] for an overview) an *operational* model and a *denotational* model. The operational model is an F-coalgebra structure on the initial Σ-algebra $(\mu\Sigma, \iota)$ and the denotational model is given by a Σ-algebra structure on the final F-coalgebra $(\nu F, t)$; we denote those structures by $c : \mu\Sigma \to F(\mu\Sigma)$ and $\alpha : \Sigma(\nu F) \to \nu F$. Notice that c is uniquely determined by the commutativity of the diagram below[5]:

$$\begin{tikzcd}
\Sigma(\mu\Sigma) \arrow[rr, "\iota"] \arrow[d, "\langle c, id\rangle"'] & & \mu\Sigma \arrow[d, "c"] \\
\Sigma(F(\mu\Sigma) \times \mu\Sigma) \arrow[r, "\lambda"'] & F(\Sigma(\mu\Sigma) + \mu\Sigma) \arrow[r, "{F[\iota, id]}"'] & F(\mu\Sigma)
\end{tikzcd} \tag{2}$$

Similarly, α is uniquely determined by the commutativity of the "dual" diagram (replacing $\mu\Sigma$ by νF and reversing and renaming arrows as appropriate).

In concrete instances, c provides behaviour on closed terms over the signature of the algebraic operations specified, and α provides the denotational semantics of the algebraic operations as specified by λ, taking input from the final coalgebra.

In the previous paper [14] we assumed that a bipointed specification $\lambda : \Sigma(F \times Id) \to F(\Sigma + Id)$ is given, where Σ is a *strongly* finitary functor [4], i. e., Σ is finitary and it preserves finitely presentable objects.

Example 2.10 (1) The class of strongly finitary functors on Set contains the identity functor, all constant functors on finite sets, the finite power-set functor $\mathcal{P}_\mathsf{f}$, and it is closed under finite products, finite coproducts and composition. A

[5] In diagrams we will omit indices of natural transformations (here λ) indicating the component.

polynomial functor Σ on Set is strongly finitary iff the corresponding signature has finitely many operation symbols of finite arity.

(2) The functor $FX = 2 \times X^A$ is strongly finitary iff A is a finite set.

(3) The type functor $FX = \mathbb{R} \times X$ of stream systems as coalgebras is finitary but not strongly so. However, if we consider F as a functor on $\mathsf{Vec}_{\mathbb{R}}$, then it is strongly finitary; in fact, for every finite dimensional real vector space V, $\mathbb{R} \times V$ is finite dimensional, too.

The main result in [14] is the following

Theorem 2.11 *Let λ be a bipointed specification where Σ is strongly finitary. Then there is a unique Σ-algebra structure $\beta : \Sigma(\rho F) \to \rho F$ such that the following diagram commutes:*

$$\begin{tikzcd}
\Sigma(\rho F) \arrow[r, "\Sigma\langle r,id\rangle"] \arrow[d, "\beta"'] & \Sigma(F(\rho F) \times \rho F) \arrow[r, "\lambda_{\rho F}"] & F(\Sigma(\rho F) + \rho F) \arrow[d, "F[\beta,id]"] \\
\rho F \arrow[rr, "r"] & & F(\rho F)
\end{tikzcd} \tag{3}$$

It then follows that the unique F-coalgebra homomorphism $(\rho F, r) \to (\nu F, t)$ is a Σ-algebra homomorphism from $(\rho F, \beta) \to (\nu F, \alpha)$. So in those cases where ρF is a subcoalgebra of νF, β is a restriction of α to ρF. This shows that the rational fixpoint is closed under operations on the denotational model specified by bipointed specifications.

In [14], we also provided a number of applications, which we briefly recall. In each case Σ is a polynomial functor for a finite signature.

Labelled transition systems. As already mentioned in the discussion above, for $FX = \mathcal{P}_{\mathsf{f}}(A \times X)$ and a polynomial endofunctor Σ on Set corresponding to a finite signature, bipointed specifications correspond precisely to transition system specifications in Aceto's simple GSOS format. As a special case of Theorem 2.11 we thus obtain the well-known result that for a finite signature, finite state processes (i. e., the elements of ρF) are closed under operations specified by simples GSOS rules. This includes for example all CCS combinators and many other operations. But the results on the simple GSOS format are not restricted to finite signatures. So one aim of the present paper is to extend our previous results to infinite signatures, and we do this in Section 4.

Streams. For the functor $FX = \mathbb{R} \times X$ and Σ a polynomial functor, we worked out a concrete rule format which is equivalent to bipointed specifications. So Theorem 2.11 yields the result that the coalgebra ρF of eventially periodic streams is closed under operations specified by rules in our format. Concrete examples include the well-known zipping operation and many others.

Non-deterministic automata. This application considers $FX = 2 \times (\mathcal{P}_{\mathsf{f}}X)^A$, and here we provide a concrete rule format that yields bipointed specifications (but not necessarily conversely). Theorem 2.11 then yields the result that the rational fixpoint ρF (of finite state branching behaviours) is closed under operations specified in our format. This includes examples such as the shuffle product. One would wish for formats defining operations on formal languages—so our results would then yield that regular languages are closed under such operations. However, if one works out what bipointed specifications mean for deterministic automata (i. e., $FX = 2 \times X^A$), then the format is not powerful enough to capture intesting operations like the shuffle product. So one aim of this paper is to work in the category Jsl in lieu of Set to obtain a more powerful format; we do this in Section 5.

Weighted transition systems. For $FX = (\mathcal{F}_{\mathbb{M}}X)^A$ we obtain a concrete rule format corresponding to bipointed specifications by restricting a general GSOS format for weighted transition system given by Klin [19]. Then Theorem 2.11 specializes to the result that the coalgebra ρF of all finite weighted transitions systems modulo weighted bisimilarity is closed by operations specified in our format.

3 Operational model and behaviour on free Σ-algebras

We will now make a first step towards proving our main result, the generalization of Aceto's theorem to mathematical operational semantics. We will prove in this section that for a bipointed specification the operational model is a locally finitely presentable coalgebra, our notion of regularity.

Actually, we will prove a more general result concerning free algebras first. In fact, we will show that the free monad on Σ lifts to a functor on $\mathsf{Coalg}_{\mathsf{lfp}}(F)$. This means that for every locally finitely presentable coalgebra (S, f) the free algebra $\hat{\Sigma}S$ of "terms in S" carries an operational model.

Assumption 3.1 *In this section we assume that $\lambda : \Sigma(F \times Id) \to F(\Sigma + Id)$ is a bipointed specification, where $F : \mathcal{A} \to \mathcal{A}$ is finitary and $\Sigma : \mathcal{A} \to \mathcal{A}$ a strongly finitary functor on the lfp category $\mathcal{A}$.*

Since Σ is (strongly) finitary, on every object X of $\mathcal{A}$ a free Σ-algebra $\hat{\Sigma}X$ exists. As proved by Barr [11], free algebras yield free monads. Indeed, $\hat{\Sigma}$ is the object assignment of a free monad on Σ. Recall from [3] the free algebra construction by which $\hat{\Sigma}X$ is obtained as the colimit of the chain

$$X \xrightarrow{\mathsf{inr}} \Sigma X + X \xrightarrow{\Sigma \mathsf{inr} + X} \Sigma(\Sigma X + X) + X \xrightarrow{\qquad} \cdots \tag{4}$$

Furthermore, it follows that as a functor $\hat{\Sigma}$ can be constructed as the colimit of the chain

$$Id \xrightarrow{\mathsf{inr}} \Sigma + Id \xrightarrow{\Sigma \mathsf{inr} + Id} \Sigma(\Sigma + Id) + Id \xrightarrow{\qquad} \cdots \tag{5}$$

More precisely, we define functors $T^n : \mathcal{A} \to \mathcal{A}$, $n < \omega$, by induction: $T^0 = Id$ and $T^{n+1} = \Sigma T^n + Id$. The connecting natural transformations are defined by $t_{0,1} = \mathsf{inr}$ and $t_{n+1,n+2} = \Sigma t_{n,n+1} + Id$. In order to prove the main result of this section further below we first need the following auxiliary property

Lemma 3.2 *The chain (5) lifts to a chain of endofunctors on* $\mathsf{Coalg}_{\mathsf{f}}(F)$*.*

Theorem 3.3 *The free monad $\hat{\Sigma} : \mathcal{A} \to \mathcal{A}$ lifts to an functor on* $\mathsf{Coalg}_{\mathsf{lfp}}(F)$*.*

Since $\mu\Sigma = \hat{\Sigma}0$, it follows that $\mu\Sigma$ carries *some* F-coalgebra structure that turns it into a locally finitely presentable coalgebra. It remains to show that the coalgebra structure on $\mu\Sigma$ provided by the previous theorem is indeed the structure $c : \mu\Sigma \to F(\mu\Sigma)$ of the operational model from the previous section:

Theorem 3.4 *The operational model of λ is an object in $\mathsf{Coalg}_{\mathsf{lfp}}(F)$.*

4 Finite dependency

With Theorem 3.4 we have the main ingredient for generalizing Aceto's theorem for simple GSOS specifications. However, notice that our restriction to strongly finitary functors Σ means that Theorem 3.4 only generalizes Aceto's theorem for the special case of transition system specifications over a *finite* signature of specified operations. However, Aceto's theorem was proved for transition system specifications having *finite dependency*. In this section we briefly recall that concept. Then we generalize finite dependency to bipointed specifications, and we prove that our previous results hold for bipointed specifications having finite dependency.

4.1 GSOS specifications having finite dependency. Let $\mathcal{T}$ be a transition system specification in the GSOS format defining operations in the signature Σ (see [2] and Section 2.3). *Operator dependency* is the smallest transitive relation on Σ such that an operation f depends on an operation g if there is a rule in $\mathcal{T}$ of the form (1) where g occurs in the term t. We say that $\mathcal{T}$ has *finite dependency* if each operation f of Σ only depends on finitely many other operations.

The *positive trigger* of a rule (1) is the sequence $\langle \{a_{ij} \mid j = 1, \ldots, m_i\} \rangle_{i=1,\ldots,\mathsf{ar}(f)}$. An operation f is called *bounded* if for every positive trigger there are only finitely many rules with f on the left-hand side of the conclusion. In the following theorem, by the associated transition system of $\mathcal{T}$ we mean the (operational) term model given by the initial Σ-algebra. Regularity means that from every state there are only finitely many other states reachable by a sequence of transitions.

Theorem 4.1 ([2, Theorem 5.28]) *Let $\mathcal{T}$ be a transition system specification in simple GSOS format having finite dependency, where every operation is bounded. Then the associated transition system of $\mathcal{T}$ is regular.*

Example 4.2 A simple example of a transition system specification is given by the prefixing operation for an infinite label alphabet A; the infinite rule set in (6) obviously has finite dependency.

$$\frac{}{a.P \xrightarrow{a} P} \quad (a \in A) \qquad (6)$$

4.2 Bipointed specifications having finite dependency. As we recalled in Section 2.3, for a finite signature, simple GSOS specifications correspond to bipointed specifications $\Sigma(\mathcal{P}_{\mathsf{f}}(A \times Id) \times Id) \to \mathcal{P}_{\mathsf{f}}(A \times (\Sigma + Id))$. Now observe that for an arbitrary signature boundedness ensures that this 1-1-correspondence still holds; the functor $\mathcal{P}_{\mathsf{f}}$ in the codomain of the bipointed specification models the finitely many transitions specified for f for each positive trigger.

Now we will analyze how finite dependency can be captured on the level of bipointed specifications. Let $\mathcal{T}$ be a transition system specification satisfying the conditions in Theorem 4.1 and let $\lambda : \Sigma(F \times Id) \to F(\Sigma + Id)$ be the corresponding bipointed specification (where Σ is a not necessarily finitary polynomial endofunctor on Set). Suppose that Γ is a subfunctor of Σ that corresponds to a subsignature that is closed under operator dependency in Σ and let $\mathsf{in}_\Gamma : \Gamma \to \Sigma$ be the corresponding inclusion map. Then there exists a bipointed specification $\lambda_\Gamma : \Gamma(F \times Id) \to F(\Gamma + Id)$ such that in_Γ is a *morphism of bipointed specifications*, i.e. the square on the right commutes. Also every inclusion $m : \Gamma \to \Gamma'$ between closed subsignatures of Σ is a morphism of bipointed specifications; one has $F(m+Id)\cdot\lambda_\Gamma = \lambda_{\Gamma'}\cdot m(F\times Id)$. Recall from Example 2.9(1) that a polynomial functor Γ is strongly finitary iff its associated signature is finite.

$$\begin{tikzcd}
\Gamma(F \times Id) \arrow[r, "\lambda_\Gamma"] \arrow[d, "\mathsf{in}_\Gamma(F\times Id)"'] & F(\Gamma + Id) \arrow[d, "F(\mathsf{in}_\Gamma + Id)"] \\
\Sigma(F \times Id) \arrow[r, "\lambda"'] & F(\Sigma + Id)
\end{tikzcd} \tag{7}$$

Proposition 4.3 *Let $\mathcal{T}$ be a transition system specification as in Theorem 4.1 and let $\lambda : \Sigma(F \times Id) \to F(\Sigma + Id)$ be its corresponding bipointed specification. Then Σ is the directed union of a diagram of strongly finitary polynomial functors Γ such that there exist λ_Γ as in (7).*

So the previous proposition states that λ is the directed union of the λ_Γ. In the following definition we consider the colimit of a filtered diagram of bipointed specifications $\lambda_\Gamma : \Gamma(F \times Id) \to F(\Gamma + Id)$, i.e. the bipointed specification for the colimit Σ of all functors Γ from the diagram uniquely determined by the commutativity of the squares (7).

Definition 4.4 Let $F : \mathcal{A} \to \mathcal{A}$ be finitary and $\Sigma : \mathcal{A} \to \mathcal{A}$. A bipointed specification $\lambda : \Sigma(F \times Id) \to F(\Sigma + Id)$ has *finite dependency* if it is the filtered colimit of a diagram of bipointed specifications $\lambda_\Gamma : \Gamma(F \times Id) \to F(\Gamma + Id)$ where each Γ is a strongly finitary functor.

Remark 4.5 (1) One common instance of the above definition is when Σ can be decomposed into a (not necessarily finite) coproduct $\Sigma = \coprod_{i\in I} \Sigma_i$ such that there are bipointed specifications $\lambda_i : \Sigma_i(F \times Id) \to F(\Sigma_i + Id)$ such that (7) commutes with Γ replaced by Σ_i for each $i \in I$. Indeed, Σ is then the filtered colimit of all $\Sigma_J = \coprod_{i\in J} \Sigma_i$, where J ranges over all finite subsets of I with λ_J formed by the obvious "copairing" involving those λ_i with $i \in J$. For a concrete example consider $FX = \mathbb{R} \times X$ and the behavioural differential equation (see [28]) $\hat{r} = r : \hat{r}$ for every $r \in \mathbb{R}$. Then one has $I = \mathbb{R}$ and Σ_i is contant on 1 for all i.

(2) That filtered colimits are necessary in Definition 4.4 is demonstrated by the usual definition of constants in the stream calculus [28]: $[r] = r : [0]$. All constants $[r]$ depend on $[0]$, and therefore the signature can not be decomposed into a coproduct. In the context of simple GSOS rules on transition systems, a similar example can be found by defining infinitely many constants c_n, $n < \omega$, by the axioms $c_{n+1} \xrightarrow{a} c_n$, for some $a \in A$. This specification cannot be decomposed into finite independent parts as in point (1) above.

Proposition 4.6 *Let λ be a bipointed specification having finite dependency, and let $(\lambda_\Gamma)_{\Gamma\in\mathcal{D}}$ be as in Definition 4.4. Then, for each Γ, the denotational models $\alpha : \Sigma(\nu F) \to \nu F$ and $\alpha_\Gamma : \Gamma(\nu F) \to \nu F$ of λ and λ_Γ, respectively, satisfy*

$$\alpha_\Gamma = (\Gamma(\nu F) \xrightarrow{\mathsf{in}_\Gamma} \Sigma(\nu F) \xrightarrow{\alpha} \nu F).$$

This proposition is related to results of Lenisa, Power and Watanabe [22, Section 5] for distributive laws of monads over copointed endofunctors. Indeed, notice that a bipointed specification can equivalently be presented as a distributive law of the free pointed functor $\Sigma + Id$ over the cofree copointed functor $F \times Id$, and the latter gives rise to a distributive law of the free monad on Σ over $F \times Id$. Lenisa, Power and Watanabe show how to combine distributive laws using coproduct; here we consider filtered colimits.

The following result extends the main result from [14] from the bipointed specifications considered in Section 2.3 to those with finite dependency.

Corollary 4.7 *Let λ be a bipointed specification having finite dependency. Then (a) there is a unique Σ-algebra structure $\beta : \Sigma(\rho F) \to \rho F$ such that the diagram (3) commutes, and (b) the unique F-coalgebra homomorphism $(\rho F, r) \to (\nu F, t)$ is a Σ-algebra homomorphism from $(\rho F, \beta)$ to $(\nu F, \alpha)$.*

We are now ready to state the main result of this paper, the generalization of Theorem 4.1 to bipointed specifications.

Theorem 4.8 *Let λ be a bipointed specification having finite dependency. Then the lifted functor $\hat{\Sigma} : \mathsf{Coalg}(F) \to \mathsf{Coalg}(F)$ restricts to $\mathsf{Coalg}_{\mathsf{lfp}}(F)$.*

In other words, for every locally finitely presentable coalgebra (S, f) the free Σ-algebra $\hat{\Sigma} S$ carries a canonical locally finitely presentable coalgebra. So finally, we obtain the desired generalization of Aceto's theorem:

Theorem 4.9 *Let λ be a bipointed specification having finite dependency. Then the operational model of λ is a locally finitely presentable coalgebra.*

Notice that this theorem is not just a trivial corollary of Theorem 4.8; as for Theorem 3.4 we still need to prove that the canonical F-coalgebra structure arising on $\hat{\Sigma} 0 = \mu\Sigma$ coincides with the operational model $c : \nu\Sigma \to F(\nu\Sigma)$.

Remark 4.10 We chose to present all our results for bipointed specifications because in applications it is easier to find concrete rule formats corresponding to them. But we believe that all of our results can be proved more generally for so-called *coGSOS laws* $\Sigma\bar{F} \to F(\Sigma + Id)$, where $\bar{F}$ denotes the cofree comonad on F (see Klin [20, Section 6.4]).

5 A rule format for operations on regular languages

In [14] there are a number of examples of concrete formats and operations corresponding to bipointed specifications. All of these examples are on Set. However, for example in the case of deterministic automata, bipointed specifications on Set are rather limited; standard operations like concatenation, Kleene star or the shuffle product of languages cannot be specified by bipointed specifications for $FX = 2 \times X^A$ on Set. But moving from Set to the category Jsl bipointed specifications allow for different and more powerful specification formats that allow to use the join semilattice operations (union and $\perp$) in the conclusion of a rule. We will now work out such a concrete specification format and then show that concatenation, Kleene star and the shuffle operations can be specified with a bipointed specification.

Recall that the functor $F = 2 \times \mathrm{Id}^A$ lifts to the functor $\bar{F} = 2 \times \mathrm{Id}^A$ on Jsl, the category of join semilattices, where $2 = \{0, 1\}$ is the join semilattice where 0 is bottom and the join is the usual "or" operation on bits. Recall from Example 2.9(2) that the rational fixpoint of $\bar{F}$ is carried by the set of regular languages as well. In this section we exploit this fact to derive a concrete format for operations on regular languages from bipointed specifications for $\bar{F}$. This format is more expressive than bipointed specifications for F, as the join semilattice structure allows to express non-determinism in the conclusion of rules.

Before we present a concrete rule format we will analyze (certain) bipointed specifications for $\bar{F}$. In the sequel let $U : \mathsf{Jsl} \underset{\longrightarrow}{\overset{\longleftarrow}{\perp}} \mathsf{Set} : \Phi$ denote the free and forgetful functor, respectively. We also denote by $J : \mathsf{FJsl} \to \mathsf{Jsl}$ the inclusion of the full subcategory given by free join semilattices. We are interested in functors $\Sigma : \mathsf{Jsl} \to \mathsf{Jsl}$ of the form $\Phi P_\Gamma U$, where $P_\Gamma : \mathsf{Set} \to \mathsf{Set}$ is a polynomial functor associated to the signature Γ. The reason for this is that Σ-algebras are precisely join semilattices A equipped with a function $P_\Gamma : UA \to UA$, i. e., for every operation symbol $\gamma \in \Gamma$ a (not necessarily join preserving) operation $A^{ar(\gamma)} \to A$.

Lemma 5.1 *Families of natural transformations*

$$\hat{\gamma} : (FUJ \times UJ)^{ar(\gamma)} \Rightarrow FU(\Sigma J + J) \qquad \gamma \in \Gamma \tag{8}$$

are in one-to-one correspondence with bipointed specifications of $\Sigma = \Phi P_\Gamma U$ over the functor $\bar{F}$.

We proceed to move from free join semilattices to plain sets and consider natural transformations

$$\bar{\gamma} : (F \times \mathrm{Id})^{ar(\gamma)} \Rightarrow FU\Phi(P_\Gamma U \Phi + \mathrm{Id}) \qquad \gamma \in \Gamma \tag{9}$$

Such families of natural transformations induce bipointed specifications, but the converse does not hold.

Lemma 5.2 *Every $\bar{\gamma}$ as in (9) induces a $\hat{\gamma}$ as in (8), and consequently such a collection induces a bipointed specification.*

Remark 5.3 The above treatment of bipointed specifications on Jsl does not depend on the specific properties of join semilattices, but works similarly for any locally finite variety.

We are now ready to define a concrete syntactic rule format, inducing the above families of natural transformations $\hat{\gamma}$.

5.1 A concrete format for deterministic automata on Jsl. In the remainder of this section let Σ be a finitary signature. A *transition rule* and an *output rule* are of the form

$$\frac{\{x_i{\downarrow}\}_{i \in I} \qquad \{x_i{\uparrow}\}_{i \in J}}{\sigma(x_1, \ldots, x_n) \xrightarrow{a} t} \qquad \text{and} \qquad \frac{\{x_i{\downarrow}\}_{i \in I} \qquad \{x_i{\uparrow}\}_{i \in J}}{\sigma(x_1, \ldots, x_n){\downarrow}}$$

respectively, where $x_1, \ldots, x_n$ is a collection of pairwise distinct variables, σ an n-ary operator of Σ; further $I, J \subseteq \{1, 2, \ldots, n\}$ and t is a term over the grammar

$$t ::= \bot \mid t \oplus t \mid \tau(u_1, \ldots, u_{ar(\tau)}) \mid x \qquad\qquad u ::= \bot \mid u \oplus u \mid x,$$

where τ ranges over the operators of Σ, x ranges over the least collection of variables V such that $x_i \in V$ for all i, and for each alphabet letter $a \in A$ and index $i \le n$ there is a distinct variable $x_i^a \in V$. Intuitively, $x_i{\uparrow}$ and $x_i{\downarrow}$ represent states that must be non-final and final, respectively, and x_i^a represents the unique state reached by x_i after an a-transition [6]. A *(bipointed) DFA (SOS) specification* is a set of transition rules and output rules such that for every operator σ of Σ, every alphabet symbol $a \in A$ and every possible sets of premises $\{x_i{\downarrow}\}_{i \in I}$ and $\{x_i{\uparrow}\}_{i \in J}$ only finitely many rules with conclusion $\sigma(x_1, \ldots, x_n) \xrightarrow{a} t$ exist. (Notice that this finiteness property corresponds to boundedness of GSOS specifications.)

Operator dependency on Σ and finite dependency of a DFA specification is defined in exactly the same way as for GSOS specifications (see Section 4.1).

Proposition 5.4 *Any DFA specification (having finite dependency) induces a bipointed specification (having finite dependency).*

Thus by Corollary 4.7 the rational fixpoint, i.e., the set of regular languages, is closed under any operations defined by a DFA specification having finite dependency. And by Theorem 4.9 the operational model is locally finite. We proceed to show several examples.

Given two words w and v, the *shuffle* of w and v, denoted $w \bowtie v$, is the set of words obtained by arbitrary interleavings of w and v [30]. For example, $ab \bowtie c = \{abc, acb, cab\}$. The shuffle of two languages L_1 and L_2 is the pointwise extension: $L_1 \bowtie L_2 = \bigcup_{w \in L_1, v \in L_2} w \bowtie v$. The shuffle operator can be defined in terms of a DFA specification as follows:

$$\frac{x \xrightarrow{a} x'}{x \bowtie y \xrightarrow{a} x' \bowtie y} \qquad \frac{y \xrightarrow{a} y'}{x \bowtie y \xrightarrow{a} x \bowtie y'} \qquad \frac{x{\downarrow} \quad y{\downarrow}}{(x \bowtie y){\downarrow}}$$

By Corollary 4.7 the set of regular languages is closed under shuffle.

Concatenation, Kleene star, a single alphabet letter and the neutral element $1 = \{\varepsilon\}$ w.r.t. concatenation, are defined as follows (the Kleene star is defined using an additional binary operation f, such that intuitively $f(L_1, L_2) = L_1 \cdot L_2^*$):

$$\frac{x \xrightarrow{a} x'}{x \cdot y \xrightarrow{a} x' \cdot y} \qquad \frac{x{\downarrow} \quad y \xrightarrow{a} y'}{x \cdot y \xrightarrow{a} y'} \qquad \frac{x{\downarrow} \quad y{\downarrow}}{(x \cdot y){\downarrow}} \qquad\qquad \frac{}{a \xrightarrow{a} 1} \quad a \in A$$

$$\frac{x \xrightarrow{a} x'}{f(x,y) \xrightarrow{a} f(x',y)} \qquad \frac{x{\downarrow} \quad y \xrightarrow{a} y'}{f(x,y) \xrightarrow{a} f(y',y)} \qquad \frac{x{\downarrow}}{f(x,y){\downarrow}} \qquad\qquad \frac{}{1{\downarrow}}$$

For the corresponding signature Γ the functor $\Sigma = FP_\Gamma U$ on Jsl thus represents syntactically the above operations, in addition to the join semilattices operations.

[6] In analogy with standard SOS we will denote x_i^a by a variable y by writing a transition $x_i \xrightarrow{a} y$ in the premise of the rule.

Thus the initial algebra of Σ consists of *regular expressions* (with a binary Kleene star) plus the join semilattice equations. So the *operational model* is precisely the coalgebra of regular expressions; by Theorem 4.9 this is locally finite. As such, we obtain for free that the number of derivatives of a regular expression is finite modulo the join semilattice equations (cf. [16]).

Interestingly, Proposition 5.4 works for any DFA specification having finite dependency, thus also when considering signature with an infinite set of operators. Consider, for example, the (obviously infinite) signature containing all regular languages $L \subseteq A^*$ as constant, together with the following DFA specification:

$$\frac{}{L \xrightarrow{a} L_a} \qquad \frac{}{L\downarrow} \quad \text{if } \varepsilon \in L$$

where L_a is the a-derivative of L given by $\{w \mid aw \in L\}$. Because every regular languages has finitely many different derivatives [16], the above DFA specification has finite dependency, and thus by Theorem 4.9 the operational model is locally finite (it coincides, in fact, with the rational fixpoint, with, as carrier, the set of all regular languages).

6 Conclusions and future work

We have generalized Aceto's theorem on the regularity of the operational model of a transition system specification from process algebra to the realm of mathematical operational semantics of Turi and Plotkin. In previous work [14] it was already shown that bipointed specifications for $F = \mathcal{P}_\mathsf{f}(A \times Id)$ generalize Aceto's simple GSOS format, and it was proved that for general bipointed specifications of a strongly finitary functor Σ over a finitary one F a canonical Σ-algebra structure is induced on the rational fixpoint of F "restricting" the denotational model on the final coalgebra for F. Here we have extended this result to finitary functors Σ that are not necessarily strongly finitary. The key to our extension is an abstract formulation of the notion of finite dependendy for bipointed specifications that captures Aceto's more concrete notion for simple GSOS specifications as a special instance. This then allows us to prove our generalisation of Aceto's result in Theorem 4.9: the operational model of such a specification is a locally finitely presentable coalgebra. The latter property is interesting for a possible tool development, as in any locally finite variety it implies decidability of bisimilarity: there are only finitely many states to check. Moreover, recent results on up-to context techniques [26] may lead to a generic and efficient construction of a bisimulation witness of the desired equivalence.

Our second contribution is the new rule format of DFA specifications for operations on formal languages. These specifications are obtained by instantiating bipointed specifications for functors of the form $\Sigma = \Phi P_\Gamma U$ on the category of join semilattices. From our results we then conclude that regular languages are closed under operations specified by DFA specification, and as a corollary we also obtain the well-known result that regular expressions have only finitely many derivatives modulo the axioms of join semilattices.

Many interesting directions are still to be explored. The process described in

Section 5.1 can easily be adapted to other locally finite varieties, allowing to derive more expressive concrete formats based on adding equations. In order to treat rational power series and even context-free ones, one needs to move to other algebraic categories, such as vector spaces and idempotent semirings. Furthermore, we plan to investigate the extension of bipointed specification to coGSOS laws [20] to allow arbitrary lookahead in premises of rules.

References

[1] Aceto, L., *GSOS and finite labelled transition systems*, Theoret. Comput. Sci. **131** (1994), pp. 181–195.

[2] Aceto, L., W. Fokkink and C. Verhoef, *Structural operational semantics*, in: *Handbook of Process Algebra*, Elsevier Science, 2001 pp. 197–292.

[3] Adámek, J., *Free algebras and automata realizations in the language of categories*, Comment. Math. Univ. Carolin. **15** (1974), pp. 589–602.

[4] Adámek, J., S. Milius and J. Velebil, *Free iterative theories: a coalgebraic view*, Math. Structures Comput. Sci. **13** (2003), pp. 259–320.

[5] Adámek, J., S. Milius and J. Velebil, *Iterative algebras at work*, Math. Structures Comput. Sci. **16** (2006), pp. 1085–1131.

[6] Adámek, J. and H.-E. Porst, *On tree coalgebras and coalgebras presentations*, Theoret. Comput. Sci. **311** (2004), pp. 257–283.

[7] Adámek, J. and J. Rosický, "Locally presentable and accessible categories," Cambridge Univ. Press, 1994.

[8] Adámek, J. and J. Rosický, *On sifted colimits and generalized varieties*, Theory Appl. Categ. **8** (2001), pp. 33–53.

[9] Adámek, J., J. Rosický and E. Vitale, "Algebraic Theories," Cambridge Univ. Press, 2011.

[10] Adámek, J. and V. Trnková, "Automata and Algebras in Categories," Mathematics and its Applications **37**, Kluwer Academic Publishers, 1990.

[11] Barr, M., *Coequalizers and free triples*, Math. Z. **116** (1970), pp. 307–322.

[12] Bartels, F., "On generalised coinduction and probabilistic specification formats," Ph.D. thesis, CWI, Amsterdam (2004).

[13] Bloom, B., S. Istrail and A. Meyer, *Bisimulation can't be traced*, J. ACM **42** (1995), pp. 232–268.

[14] Bonsangue, M., S. Milius and J. Rot, *On the specification of operations on the rational behaviour of systems*, in: *EXPRESS/SOS'12*, Elect. Proc. of Theoret. Comput. Sci. **89**, 2012, pp. 3–18.

[15] Bonsangue, M., S. Milius and A. Silva, *Sound and complete axiomatizations of coalgebraic language equivalence*, ACM Trans. Comput. Log. **14** (2013).

[16] Brzozowski, J. A., *Derivatives of regular expressions*, J. ACM **11** (1964), pp. 481–494.

[17] Gabriel, P. and F. Ulmer, "Lokal präsentierbare Kategorien," Lecture Notes Math. **221**, Springer, 1971.

[18] Johnstone, P., "Stone Spaces," Cambridge University Press, 1986.

[19] Klin, B., *Structural operational semantics for weighted transition systems*, in: *Semantics and Algebraic Specification*, LNCS **5700** (2009).

[20] Klin, B., *Bialgebras for structural operational semantics: An introduction*, Theoret. Comput. Sci. **412** (2011), pp. 5043–5069.

[21] Kurz, A. and J. Rosický, *Strongly complete logic for coalgebras*, Log. Meth. in Comput. Sci. **8** (2012), pp. 3–14.

[22] Lenisa, M., A. J. Power and H. Watanabe, *Category theory for operational semantics*, Theoret. Comput. Sci. **327** (2004).

[23] Makkai, M. and R. Paré, "Accessible categories: the foundation of categorical model theory," Contemporary Math. **104**, Amer. Math. Soc., Providence, RI, 1989.

[24] Milius, S., *A sound and complete calculus for finite stream circuits*, in: *Proc. of LICS 2010* (2010), pp. 421–430.

[25] Milner, R., "Communication and Concurrency," Prentice Hall, 1989.

[26] Rot, J. Bonsangue, M., Rutten, J., *Coalgebraic Bisimulation-Up-To*, in Proc. of SOFSEM 2013, Lecture Notes in Comput. Sci. 7741 , Springer (2013) pp. 369–381.

[27] Rutten, J., *Universal coalgebra: a theory of systems*, Theoret. Comput. Sci. **249** (2000).

[28] Rutten, J., *A coinductive calculus of streams*, Math. Structures Comput. Sci. **15** (2005), pp. 93–147.

[29] Rutten, J., *Rational streams coalgebraically*, Log. Meth. in Comput. Sci. **4** (2008), pp. 1–22.

[30] Shallit, J., "A Second Course in Formal Languages and Automata Theory," Cambridge Univ. Press, 2008.

[31] Turi, D. and G. Plotkin, *Towards a mathematical operational semantics*, in: *Proc. of LICS 1997* (1997), pp. 280–291.

A Categorical Theory of Patches

Samuel Mimram and Cinzia Di Giusto

CEA, LIST

Abstract

When working with distant collaborators on the same documents, one often uses a version control system, which is a program tracking the history of files and helping importing modifications brought by others as patches. The implementation of such a system requires to handle lots of situations depending on the operations performed by users on files, and it is thus difficult to ensure that all the corner cases have been correctly addressed. Here, instead of verifying the implementation of such a system, we adopt a complementary approach: we introduce a theoretical model, which is defined abstractly by the universal property that it should satisfy, and work out a concrete description of it. We begin by defining a category of files and patches, where the operation of merging the effect of two coinitial patches is defined by pushout. Since two patches can be incompatible, such a pushout does not necessarily exist in the category, which raises the question of which is the correct category to represent and manipulate files in conflicting state. We provide an answer by investigating the free completion of the category of files under finite colimits, and give an explicit description of this category: its objects are finite sets labeled by lines equipped with a transitive relation and morphisms are partial functions respecting labeling and relations.

1 Introduction

It is common nowadays, when working with distant collaborators on the same files (multiple authors writing an article together for instance), to use a program which will track the history of files and handle the operation of importing modifications of other participants. These software called *version control systems* (VCS for short), like `git` or Darcs, implement two main operations. When a user is happy with the changes it has brought to the files it can record those changes in a *patch* (a file coding the differences between the current version and the last recorded version) and *commit* them to a server, called a *repository*. The user can also *update* its current version of the file by importing new patches added by other users to the repository and applying the corresponding modifications to the files. One of the main difficulties to address here is that there is no global notion of "time": patches are only partially ordered. For instance consider a repository with one file A and two users u_1 and u_2. Suppose that u_1 modifies file A into B by committing a patch f, which is then imported by u_2, and then u_1 and u_2 concurrently modify the file B into C (resp. D) by committing a patch g (resp. h). The evolution of the file is

[1] This work was partially supported by the French project ANR-11-INSE-0007 REVER.

depicted on the left and the partial ordering of patches in the middle:

$$\begin{tikzcd}
C & & D \\
& B \arrow[ul, "g"] \arrow[ur, "h"'] & \\
& A \arrow[u, "f"] &
\end{tikzcd}
\qquad
\begin{tikzcd}
g & & h \\
& f \arrow[ul] \arrow[ur] &
\end{tikzcd}
\qquad
\begin{tikzcd}
& E & \\
C \arrow[ur, "h/g"] & & D \arrow[ul, "g/h"'] \\
& B \arrow[ul, "g"] \arrow[ur, "h"'] & \\
& A \arrow[u, "f"] &
\end{tikzcd}$$

Now, suppose that u_2 imports the patch g or that u_1 imports the patch h. Clearly, this file resulting from the *merging* of the two patches should be the same in both cases, call it E. One way to compute this file, is to say that there should be a patch h/g, the *residual* of h after g, which transforms C into E and has the "same effect" as h once g has been applied, and similarly there should be a patch g/h transforming D into E. Thus, after each user has imported changes from the other, the evolution of the file is as pictured on the right above. In this article, we introduce a category $\mathcal{L}$ whose objects are files and morphisms are patches. Since residuals should be computed in the most general way, we formally define them as the arrows of pushout cocones, i.e. the square in the figure on the right should be a pushout.

However, as expected, not every pair of coinitial morphisms have a pushout in the category $\mathcal{L}$: this reflects the fact that two patches can be conflicting (for instance if two users modify the same line of a file). Representing and handling such conflicts in a coherent way is one of the most difficult part of implementing a VCS (as witnessed for instance by the various proposals for Darcs: mergers, conflictors, graphictors, etc. [10]). In order to be able to have a representation for all conflicting files, we investigate the free completion of the category $\mathcal{L}$ under all pushouts, this category being denoted $\mathcal{P}$, which corresponds to adding all conflicting files to the category, in the most general way as possible. This category can easily be shown to exist for general abstract reasons, and one of the main contributions of this work is to provide an explicit description by applying the theory of presheaves. This approach paves the way towards the implementation of a VCS whose correctness is deduced from universal categorical properties.

Related work. The Darcs community has investigated a formalization of patches based on commutation properties [10]. *Operational transformations* tackle essentially the same issues by axiomatizing the notion of residual patches [9]. In both cases, the fact that residual should form a pushout cocone is never explicitly stated, excepting in informal sentences saying that "g/f should have the same effect as g once f has been applied". We should also mention another interesting approach to the problem using inverse semigroups in [4]. Finally, Houston has proposed a category with pushouts, similar to ours, in order to model conflicting files [3], see Section 6.

Plan of the paper. We begin by defining a category $\mathcal{L}$ of files and patches in Section 2. Then, in Section 3, we abstractly define the category $\mathcal{P}$ of conflicting files obtained by free finite cocompletion. Section 4 provides a concrete description of the construction in the simpler case where patches can only insert lines. We give some concrete examples in Section 5 and adapt the framework to the general case in Section 6. We conclude in Section 7.

2 Categories of files and patches

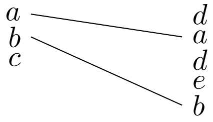
In this section, we investigate a model for a simplified VCS: it handles only one file and the only allowed operations are insertion and deletion of lines (modification of a line can be encoded by a deletion followed by an insertion). We suppose fixed a set $L = \{a, b, \ldots\}$ of *lines* (typically words over an alphabet of characters). A *file* A is a finite sequence of lines, which will be seen as a function $A : [n] \to L$ for some number of lines $n \in \mathbb{N}$, where the set $[n] = \{0, 1, \ldots, n-1\}$ indexes the lines of the files. For instance, a file A with three lines such that $A(0) = a$, $A(1) = b$ and $A(2) = c$ models the file *abc*. Given $a \in L$, we sometimes simply write a for the file $A : [1] \to L$ such that $A(0) = a$. A morphism between two files $A : [m] \to L$ and $B : [n] \to L$ is an injective increasing partial function $f : [m] \to [n]$ such that $\forall i \in [m], B \circ f(i) = A(i)$ whenever $f(i)$ is defined. Such a morphism is called a *patch*.

Definition 2.1 The category $\mathcal{L}$ has files as objects and patches as morphisms.

Notice that the category $\mathcal{L}$ is strictly monoidal with $[m] \otimes [n] = [m+n]$ and for every file $A : [m] \to L$ and $B : [n] \to L$, $(A \otimes B)(i) = A(i)$ if $i < m$ and $(A \otimes B)(i) = B(i-m)$ otherwise, the unit being the empty file $I : [0] \to L$, and tensor being defined on morphisms in the obvious way. The following proposition shows that patches are generated by the operations of inserting and deleting a line:

Proposition 2.2 *The category $\mathcal{L}$ is the free monoidal category containing L as objects and containing, for every line $a \in L$, morphisms $\eta_a : I \to a$ (insertion of a line a) and $\varepsilon_a : a \to I$ (deletion of a line a) such that $\varepsilon_a \circ \eta_a = \mathrm{id}_I$ (deleting an inserted line amounts to do nothing).*

Example 2.3 The patch corresponding to transforming the file *abc* into *dadeb*, by deleting the line c and inserting the lines labeled by d and e, is modeled by the partial function $f : [3] \to [5]$ such that $f(0) = 1$ and $f(1) = 4$ and $f(2)$ is undefined. Graphically,



The deleted line is the one on which f is not defined and the inserted lines are those which are not in the image of f. In other words, f keeps track of the unchanged lines.

In order to increase readability, we shall consider the particular case where L is reduced to a single element. In this *unlabeled* case, the objects of $\mathcal{L}$ can be identified with integers (the labeling function is trivial), and Proposition 2.2 can be adapted to achieve the following description of the category, see also [6].

Proposition 2.4 *If L is reduced to a singleton, the category $\mathcal{L}$ is the free category whose objects are integers and morphisms are generated by $s_i^n : n \to n+1$ and*

$d_i^n : n+1 \to n$ for every $n \in \mathbb{N}$ and $i \in [n+1]$ (respectively corresponding to insertion and deletion of a line at i-th position), subject to the relations

$$s_i^{n+1} s_j^n = s_{j+1}^{n+1} s_i^n \qquad d_i^n s_i^n = \mathrm{id}_n \qquad d_i^n d_j^{n+1} = d_j^n d_{i+1}^{n+1} \tag{1}$$

whenever $0 \leq i \leq j < n$.

We will also consider the subcategory $\mathcal{L}^+$ of $\mathcal{L}$, with same objects, and *total* injective increasing functions as morphisms. This category models patches where the only possible operation is the insertion of lines: Proposition 2.2 can be adapted to show that $\mathcal{L}^+$ is the free monoidal category containing morphisms $\eta_a : I \to a$ and, in the unlabeled case, Proposition 2.4 can be similarly adapted to show that it is the free category generated by morphisms $s_i^n : n \to n+1$ satisfying $s_i^{n+1} s_j^n = s_{j+1}^{n+1} s_i^n$ with $0 \leq i \leq j < n$.

3 Towards a category of conflicting files

Suppose that A is a file which is edited by two users, respectively applying patches $f_1 : A \to A_1$ and $f_2 : A \to A_2$ to the file. For instance,

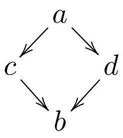
$$a\ c\ c\ b \quad \xleftarrow{f_1} \quad a\ b \quad \xrightarrow{f_2} \quad a\ b\ c\ d \tag{2}$$

Now, each of the two users imports the modification from the other one. The resulting file, after the import, should be the smallest file containing both modifications on the original file: *accbcd*. It is thus natural to state that it should be a pushout of the diagram (2). Now, it can be noticed that not every diagram in $\mathcal{L}$ has a pushout. For instance, the diagram

$$a\ c\ b \quad \xleftarrow{f_1} \quad a\ b \quad \xrightarrow{f_2} \quad a\ d\ b \tag{3}$$

does not admit a pushout in $\mathcal{L}$. In this case, the two patches f_1 and f_2 are said to be *conflicting*.

In order to represent the state of files after applying two conflicting patches, we investigate the definition of a category $\mathcal{P}$ which is obtained by completing the category $\mathcal{L}$ under all pushouts. Since, this completion should also contain an initial object (i.e. the empty file), we are actually defining the category $\mathcal{P}$ as the free completion of $\mathcal{L}$ under finite colimits: recall that a category is finitely cocomplete (has all finite colimits) if and only if it has an initial object and is closed under pushouts [6]. Intuitively, this category is obtained by adding files whose lines are not linearly ordered, but only partially ordered, such as on the left of



```
a
<<<<<<< HEAD
c
=======
d
>>>>>>> 5c55...
b
```

(4)

which would intuitively model the pushout of the diagram (3) if it existed, indicating that the user has to choose between c and d for the second line. Notice the similarities with the corresponding textual notation in `git` on the right. The name of the category $\mathcal{L}$ reflects the facts that its objects are files whose lines are linearly ordered, whereas the objects of $\mathcal{P}$ can be thought as files whose lines are only partially ordered. More formally, the category is defined as follows.

Definition 3.1 The category $\mathcal{P}$ is the *free finite conservative cocompletion* of $\mathcal{L}$: it is (up to equivalence of categories) the unique finitely cocomplete category together with an embedding functor $y : \mathcal{L} \to \mathcal{P}$ preserving finite colimits, such that for every finitely cocomplete category $\mathcal{C}$ and functor $F : \mathcal{L} \to \mathcal{C}$ preserving finite colimits, there exists, up to unique isomorphism, a unique functor $\tilde{F} : \mathcal{P} \to \mathcal{C}$ preserving finite colimits and satisfying $\tilde{F} \circ y = F$:

$$\begin{tikzcd}
\mathcal{L} \arrow[r, "F"] \arrow[d, "y"'] & \mathcal{C} \\
\mathcal{P} \arrow[ur, dotted, "\tilde{F}"'] &
\end{tikzcd}$$

Above, the term *conservative* refers to the fact that we preserve colimits which already exist in $\mathcal{L}$ (we will only consider such completions here). The “standard” way to characterize the category $\mathcal{P}$, which always exists, is to use the following folklore theorem, often attributed to Kelly [5,1]:

Theorem 3.2 *The conservative cocompletion of the category $\mathcal{L}$ is equivalent to the full subcategory of $\hat{\mathcal{L}}$ whose objects are presheaves which preserve finite limits, i.e. the image of a limit in $\mathcal{L}^{\mathrm{op}}$ (or equivalently a colimit in $\mathcal{L}$) is a limit in* **Set** *(and limiting cones are transported to limiting cones). The finite conservative cocompletion $\mathcal{P}$ can be obtained by further restricting to presheaves which are finite colimits of representables.*

Example 3.3 The category **FinSet** of finite sets and functions is the conservative cocompletion of the terminal category $\mathbf{1}$.

We recall that the category $\hat{\mathcal{L}}$ of *presheaves* over $\mathcal{L}$, is the category of functors $\mathcal{L}^{\mathrm{op}} \to$ **Set** and natural transformations between them. The *Yoneda functor* $y : \mathcal{L} \to \hat{\mathcal{L}}$ defined on objects $n \in \mathcal{L}$ by $yn = \mathcal{L}(-, n)$, and on morphisms by postcomposition, provides a full and faithful embedding of $\mathcal{L}$ into the corresponding presheaf category, and can be shown to corestrict into a functor $y : \mathcal{L} \to \mathcal{P}$ [1]. A presheaf of the form yn for some $n \in \mathcal{L}$ is called *representable*.

Extracting a concrete description of the category $\mathcal{P}$ from the above proposition is a challenging task, because we a priori need to characterize firstly all diagrams admitting a colimit in $\mathcal{L}$, and secondly all presheaves in $\hat{\mathcal{L}}$ which preserve those diagrams. This paper introduces a general methodology to build such a category. In particular, perhaps a bit surprisingly, it turns out that we have to “allow cycles” in the objects of the category $\mathcal{P}$, which will be described as *the category whose objects are finite sets labeled by lines together with a transitive relation and morphisms are partial functions respecting labels and relations.*

4 A cocompletion of files and insertions of lines

In order to make our presentation clearer, we shall begin our investigation of the category $\mathcal{P}$ in a simpler case, which will be generalized in Section 6: we compute the free finite cocompletion of the category $\mathcal{L}^+$ (patches can only insert lines) in the case where the set of labels is a singleton. *To further lighten notations, in this section, we simply write $\mathcal{L}$ for this category.*

We sometimes characterize the objects in $\mathcal{L}$ as finite colimits of objects in a subcategory $\mathcal{G}$ of $\mathcal{L}$. This category $\mathcal{G}$ is the full subcategory of $\mathcal{L}$ whose objects are 1 and 2: it is the free category on the graph $1 \rightrightarrows 2$, the two arrows being s_0^1 and s_1^1. The category $\hat{\mathcal{G}}$ of presheaves over $\mathcal{G}$ is the category of *graphs*: a presheaf $P \in \hat{\mathcal{G}}$ is a graph with $P(1)$ as vertices, $P(2)$ as edges, the functions $P(s_1^1)$ and $P(s_0^1)$ associate to a vertex its source and target respectively, and morphisms correspond to usual morphisms of graphs. We denote by $x \twoheadrightarrow y$ a path going from a vertex x to a vertex y in such a graph. The inclusion functor $I : \mathcal{G} \to \mathcal{L}$ induces, by precomposition, a functor $I^* : \hat{\mathcal{L}} \to \hat{\mathcal{G}}$. The image of a presheaf in $\hat{\mathcal{L}}$ under this functor is called its *underlying graph*. By well known results about presheaves categories, this functor admits a right adjoint $I_* : \hat{\mathcal{G}} \to \hat{\mathcal{L}}$: given a graph $G \in \hat{\mathcal{G}}$, its image under the right adjoint is the presheaf $G_* \in \hat{\mathcal{L}}$ such that for every $n \in \mathbb{N}$, $G_*(n+1)$ is the set of paths of length n in the graph G, with the expected source maps, and $G_*(0)$ is reduced to one element.

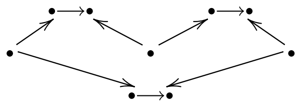
Recall that every functor $F : \mathcal{C} \to \mathcal{D}$ induces a *nerve functor* $N_F : \mathcal{D} \to \hat{\mathcal{C}}$ defined on an object $A \in \mathcal{C}$ by $N_F(A) = \mathcal{D}(F-, A)$ [7]. Here, we will consider the nerve $N_I : \mathcal{L} \to \hat{\mathcal{G}}$ associated to the inclusion functor $I : \mathcal{G} \to \mathcal{L}$. An easy computation shows that the image $N_I(n)$ of $n \in \mathcal{L}$ is a graph with n vertices, so that its objects are isomorphic to $[n]$, and there is an arrow $i \to j$ for every $i, j \in [n]$ such that $i < j$. For instance,

$$N_I(3) = \begin{tikzcd} 0 \arrow[r] \arrow[rr, bend right] & 1 \arrow[r] & 2 \end{tikzcd} \qquad\qquad N_I(4) = \begin{tikzcd} 0 \arrow[r] \arrow[rr, bend left] \arrow[rrr, bend right] & 1 \arrow[r] \arrow[rr, bend left] & 2 \arrow[r] & 3 \end{tikzcd}$$

It is, therefore, easy to check that this embedding is full and faithful, i.e. morphisms in $\mathcal{L}$ correspond to natural transformations in $\hat{\mathcal{G}}$. Moreover, since $N_I(1)$ is the graph reduced to a vertex and $N_I(2)$ is the graph reduced to two vertices and one arrow between them, every graph can be obtained as a finite colimit of the graphs $N_I(1)$ and $N_I(2)$ by "gluing arrows along vertices". For instance, the initial graph $N_I(0)$ is the colimit of the empty diagram, and the graph $N_I(3)$ is the colimit of the diagram

$$\begin{tikzcd}
 & N_I(2) & & N_I(2) & \\
N_I(1) \arrow[ur, "N_I(s_1)"] \arrow[drr, "N_I(s_1)"'] & & N_I(1) \arrow[ul, "N_I(s_0)"'] \arrow[ur, "N_I(s_1)"] & & N_I(1) \arrow[ul, "N_I(s_0)"'] \arrow[dll, "N_I(s_0)"] \\
 & & N_I(2) & &
\end{tikzcd}$$

which may also be drawn as on the left of



$$\begin{tikzcd}
 & 2 & & 2 & \\
1 \arrow[ur] \arrow[dr] & & 1 \arrow[ul] \arrow[ur] & & 1 \arrow[ul] \arrow[dl] \\
 & 2 & & &
\end{tikzcd}$$

by drawing the graphs $N_I(0)$ and $N_I(1)$. Notice, that the object 3 is the colimit of the corresponding diagram in $\mathcal{L}$ (on the right), and this is generally true for all objects of $\mathcal{L}$, moreover this diagram is described by the functor $\mathrm{El}(N_I(3)) \xrightarrow{\pi} \mathcal{L}$. The notation $\mathrm{El}(P)$ refers to the *category of elements* of a presheaf $P \in \hat{\mathcal{C}}$, whose objects are pairs (A, p) with $A \in \mathcal{C}$ and $p \in P(A)$ and morphisms $f : (A, p) \to (B, q)$ are morphisms $f : A \to B$ in $\mathcal{C}$ such that $P(f)(q) = p$, and π is the first projection functor. The functor $I : \mathcal{G} \to \mathcal{L}$ is thus a dense functor in the sense of Definition 4.2 below, see [7] for details.

Proposition 4.1 *Given a functor $F : \mathcal{C} \to \mathcal{D}$, with $\mathcal{D}$ cocomplete, the associated nerve $N_F : \mathcal{D} \to \hat{\mathcal{C}}$ admits a left adjoint $R_F : \hat{\mathcal{C}} \to \mathcal{D}$ called the* realization *along F. This functor is defined on objects $P \in \hat{\mathcal{C}}$ by*

$$R_F(P) \quad = \quad \mathrm{colim}(\mathrm{El}(P) \xrightarrow{\pi} \mathcal{C} \xrightarrow{F} \mathcal{D})$$

Proof Given a presheaf $P \in \hat{\mathcal{C}}$ and an object D, it can be checked directly that morphisms $P \to N_F D$ in $\hat{\mathcal{C}}$ with cocones from $\mathrm{El}(P) \xrightarrow{\mathcal{D}}$ to D, which in turn are in bijection with morphisms $R_F(P) \to D$ in $\mathcal{D}$, see [7]. □

Definition 4.2 A functor $F : \mathcal{C} \to \mathcal{D}$ is *dense* if it satisfies one of the two equivalent conditions:

(i) the associated nerve functor $N_F : \mathcal{D} \to \hat{\mathcal{C}}$ is full and faithful,

(ii) every object of $\mathcal{D}$ is canonically a colimit of objects in $\mathcal{C}$: for every $D \in \mathcal{D}$,

$$D \quad \cong \quad \mathrm{colim}(\mathrm{El}(N_F D) \xrightarrow{\pi} \mathcal{C} \xrightarrow{F} \mathcal{D}) \tag{5}$$

Since the functor I is dense, every object of $\mathcal{L}$ is a finite colimit of objects in $\mathcal{G}$, and $\mathcal{G}$ does not have any non-trivial colimit. One could expect the free conservative finite cocompletion of $\mathcal{L}$ to be the free finite cocompletion $\mathcal{P}$ of $\mathcal{G}$. We will see that this is not the case because the image in $\mathcal{L}$ of a non-trivial diagram in $\mathcal{G}$ might still have a colimit. By Theorem 3.2, the category $\mathcal{P}$ is the full subcategory of $\hat{\mathcal{L}}$ of presheaves preserving limits, which we now describe explicitly. This category will turn out to be equivalent to a full subcategory of $\hat{\mathcal{G}}$ (Theorem 4.8). We should first remark that those presheaves satisfy the following properties:

Proposition 4.3 *Given a presheaf $P \in \hat{\mathcal{L}}$ which is an object of $\mathcal{P}$,*

(i) *the underlying graph of P is finite,*

(ii) *for each non-empty path $x \twoheadrightarrow y$ there exists exactly one edge $x \to y$ (in particular there is at most one edge between two vertices),*

(iii) *$P(n+1)$ is the set of paths of length n in the underlying graph of P, and $P(0)$ is reduced to one element.*

Proof We suppose given a presheaf $P \in \mathcal{P}$, it preserves limits by Theorem 3.2.

The diagram on the left

$$\begin{tikzcd}[column sep=small]
& 3 & & & & P(3) \arrow[dl, "P(s_2^2)"'] \arrow[dr, "P(s_0^2)"] & \\
2 \arrow[ur, "s_2^2"] & & 2 \arrow[ul, "s_0^2"'] & & P(2) \arrow[dr, "P(s_0^1)"'] & & P(2) \arrow[dl, "P(s_1^1)"] \\
& 1 \arrow[ul, "s_0^1"] \arrow[ur, "s_1^1"'] & & & & P(1) &
\end{tikzcd}$$

is a pushout in $\mathcal{L}$, or equivalently the dual diagram is a pullback in $\mathcal{L}^{\mathrm{op}}$. Therefore, writing D for the diagram $2\xleftarrow{s_0^1}1\xrightarrow{s_1^1}2$ in $\mathcal{L}$, a presheaf $P \in \mathcal{P}$ should satisfy $P((\operatorname{colim} D)^{\mathrm{op}}) \cong \lim P(D^{\mathrm{op}})$, i.e. the above pushout diagram in $\mathcal{L}$ should be transported by P into the pullback diagram in **Set** depicted on the right of the above figure. This condition can be summarized by saying that P should satisfy the isomorphism $P(3) \cong P(2) \times_{P(1)} P(2)$ (and this isomorphism should respect obvious source and target maps given by the fact that the functor P should send a limiting cone to a limiting cone). From this fact, one can deduce that the elements α of $P(3)$ are in bijection with the paths $x \to y \to z$ of length 2 in the underlying graph of P going from $x = P(s_2^2 s_1^1)(\alpha)$ to $z = P(s_0^2 s_0^1)(\alpha)$. In particular, this implies that for any path $\alpha = x \to y \to z$ of length 2 in the underlying graph of P, there exists an edge $x \to z$, which is $P(s_1^2)(\alpha)$. More generally, given any integer $n > 1$, the object $n + 1$ is the colimit in $\mathcal{L}$ of the diagram

$$\begin{tikzcd}[column sep=small]
& 2 & & 2 & & & & 2 & & 2 & \\
1 \arrow[ur, "s_1^1"] & & 1 \arrow[ul, "s_0^1"'] \arrow[ur, "s_1^1"] & & {} \arrow[ul, "s_0^1"'] & \cdots & {} \arrow[ur, "s_1^1"] & & 1 \arrow[ul, "s_0^1"'] \arrow[ur, "s_1^1"] & & 1 \arrow[ul, "s_0^1"']
\end{tikzcd} \tag{6}$$

with $n+1$ occurrences of the object 1, and n occurrences of the object 2. Therefore, for every $n \in \mathbb{N}$, $P(n + 1)$ is isomorphic to the set of paths of length n in the underlying graph. Moreover, since the diagram

$$\begin{tikzcd}[column sep=small]
& 2 & & 2 & & & & 2 & & 2 & \\
1 \arrow[ur, "s_1^1"] \arrow[drrrrr, "s_1^1"'] & & 1 \arrow[ul, "s_0^1"'] \arrow[ur, "s_1^1"] & & {} \arrow[ul, "s_0^1"'] & \cdots & {} \arrow[ur, "s_1^1"] & & 1 \arrow[ul, "s_0^1"'] \arrow[ur, "s_1^1"] & & 1 \arrow[ul, "s_0^1"'] \arrow[dlllll, "s_0^1"] \\
& & & & & 2 & & & & &
\end{tikzcd} \tag{7}$$

with $n + 1$ occurrences of the object 1 also admits the object $n + 1$ as colimit, we should have $P(n + 1) \cong P(n + 1) \times P(2)$ between any two vertices x and y, i.e. for every non-empty path $x \twoheadrightarrow y$ there exists exactly one edge $x \to y$. Also, since the object 0 is initial in $\mathcal{L}$, it is the colimit of the empty diagram. The set $P(0)$ should thus be the terminal set, i.e. reduced to one element. Finally, since I is dense, P should be a finite colimit of the representables $N_I(1)$ and $N_I(2)$, the set $P(1)$ is necessarily finite, as well as the set $P(2)$ since there is at most one edge between two vertices. □

Conversely, we wish to show that the conditions mentioned in the above proposition exactly characterize the presheaves in $\mathcal{P}$ among those in $\hat{\mathcal{L}}$. In order to prove so, by Theorem 3.2, we have to show that presheaves P satisfying these conditions preserve finite limits in $\mathcal{L}$, i.e. that for every finite diagram $D : \mathcal{J} \to \mathcal{L}$ admitting a colimit we have $P(\operatorname{colim} D) \cong \lim(P \circ D^{\mathrm{op}})$. It seems quite difficult to characterize the diagrams admitting a colimit in $\mathcal{L}$, however the following lemma shows that it is enough to check diagrams "generated" by a graph which admits a colimit.

Lemma 4.4 *A presheaf $P \in \hat{\mathcal{L}}$ preserves finite limits if and only if it sends the colimits of diagrams of the form*

$$\mathrm{El}(G) \xrightarrow{\pi_G} \mathcal{G} \xrightarrow{I} \mathcal{L} \tag{8}$$

to limits in **Set***, where $G \in \hat{\mathcal{G}}$ is a finite graph such that the above diagram admits a colimit. Such a diagram in $\mathcal{L}$ is said to be* generated by the graph G.

Proof In order to check that a presheaf $P \in \hat{\mathcal{L}}$ preserves finite limits, we have to check that it sends colimits of finite diagrams in $\mathcal{L}$ *which admit a colimit* to limits in **Set**, and therefore we have to characterize diagrams which admit colimits in $\mathcal{L}$. Suppose given a diagram $K : \mathcal{J} \to \mathcal{L}$. Since I is dense, every object of linear is a colimit of a diagram involving only the objects 1 and 2 (see Definition 4.2). We can therefore suppose that this is the case in the diagram K. Finally, it can be shown that diagram K admits the same colimits as a diagram containing only s_0^1 and s_1^1 as arrows (these are the only non-trivial arrows in $\mathcal{L}$ whose source and target are 1 or 2), in which every object 2 is the target of exactly one arrow s_0^1 and one arrow s_1^1. For instance, the diagram in $\mathcal{L}$ below on the left admits the same colimits as the diagram in the middle.

$$\begin{tikzcd}
2 & & 3 & & \\
& 1 \arrow[ul, "s_0^1"] \arrow[ur, "s_2^2 s_1^1"] \arrow[dr, "s_1^1"'] & & 1 \arrow[ul, "s_0^2 s_0^1"'] \arrow[dl, "s_0^1"] & 1 \arrow[dll, bend left, "s_0^1"] \\
& & 2 & &
\end{tikzcd}
\qquad
\begin{tikzcd}
& 2 & & 2 & & 2 & \\
1 \arrow[ur, "s_1^1"] & & 1 \arrow[ul, "s_0^1"'] \arrow[ur, "s_1^1"] \arrow[dr, "s_1^1"'] & & 1 \arrow[ul, "s_0^1"'] \arrow[ur, "s_1^1"] & & 1 \arrow[ul, "s_0^1"'] \arrow[dlll, "s_0^1"] \\
& & & 2 & & &
\end{tikzcd}
\qquad
\begin{tikzcd}
0 \arrow[r] & 1 \arrow[r] \arrow[rr, bend right] & 2 \arrow[r] & 3
\end{tikzcd}$$

Any such diagram K is obtained by gluing a finite number of diagrams of the form $1 \xrightarrow{s_1^1} 2 \xleftarrow{s_0^1} 1$ along objects 1, and is therefore of the form $\mathrm{El}(G) \xrightarrow{\pi} \mathcal{G} \xrightarrow{I} \mathcal{L}$ for some finite graph $G \in \hat{\mathcal{G}}$: the objects of G are the objects 1 in K, the edges of G are the objects 2 in K and the source and target of an edge 2 are respectively given by the sources of the corresponding arrows s_1^1 and s_0^1 admitting it as target. For instance, the diagram in the middle above is generated by the graph on the right. The fact that every diagram is generated by a presheaf (is a discrete fibration) also follows more abstractly and generally from the construction of the comprehensive factorization system on **Cat** [8,11]. □

Among diagrams generated by graphs, those admitting a colimit can be characterized using the following proposition:

Lemma 4.5 *Given a graph $G \in \hat{\mathcal{G}}$, the associated diagram (8) admits a colimit in $\mathcal{L}$ if and only if there exists $n \in \mathcal{L}$ and a morphism $f : G \to N_I n$ in $\hat{\mathcal{L}}$ such that every morphism $g : G \to N_I m$ in $\hat{\mathcal{L}}$, with $m \in \mathcal{L}$, factorizes uniquely through $N_I n$:*

$$\begin{tikzcd}
G \arrow[r, "f"] \arrow[rr, bend right, "g"'] & N_I n \arrow[r, dotted] & N_I m
\end{tikzcd}$$

Proof Follows from the existence of a partially defined left adjoint to N_I, in the sense of [8], given by the fact that I is dense (see Definition 4.2). □

We finally arrive at the following concrete characterization of diagrams admitting colimits:

Lemma 4.6 *A finite graph $G \in \hat{\mathcal{G}}$ induces a diagram (8) in $\mathcal{L}$ which admits a colimit if and only if it is "tree-shaped", i.e. it is*

(i) *acyclic: for any vertex x, the only path $x \twoheadrightarrow x$ is the empty path,*

(ii) *connected: for any pair of vertices x and y there exists a path $x \twoheadrightarrow y$ or a path $y \twoheadrightarrow x$.*

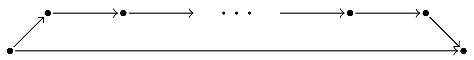
Proof Given an object $n \in \mathcal{L}$, recall that $N_I n$ is the graph whose objects are elements of $[n]$ and there is an arrow $i \to j$ if and only if $i < j$. Given a finite graph G, morphisms $f : G \to N_I n$ are therefore in bijection with functions $f : V_G \to [n]$, where V_G denotes the set of vertices of G, such that $f(x) < f(y)$ whenever there exists an edge $x \to y$ (or equivalently, there exists a non-empty path $x \twoheadrightarrow y$).

Consider a finite graph $G \in \hat{\mathcal{G}}$, by Lemma 4.5, it induces a diagram (8) admitting a colimit if there is a universal arrow $f : G \to N_I n$ with $n \in \mathcal{L}$. From this it follows that the graph is acyclic: otherwise, we would have a non-empty path $x \twoheadrightarrow x$ for some vertex x, which would imply $f(x) < f(x)$. Similarly, suppose that G is a graph with vertices x and y such that there is no path $x \twoheadrightarrow y$ or $y \twoheadrightarrow x$, and there is an universal morphism $f : G \to N_I n$ for some $n \in \mathcal{L}$. Suppose that $f(x) \leq f(y)$ (the case where $f(y) \leq f(x)$ is similar). We can define a morphism $g : G \to N_I(n+1)$ by $g(z) = f(z)+1$ if there is a path $x \twoheadrightarrow z$, $g(y) = f(x)$ and $g(z) = f(z)$ otherwise. This morphism is easily checked to be well-defined. Since we always have $f(x) \leq f(y)$ and $g(x) > g(y)$, there is no morphism $h : N_I n \to N_I(n+1)$ such that $h \circ f = g$.

Conversely, given a finite acyclic connected graph G, the relation $\leq$ defined on morphisms by $x \leq y$ whenever there exists a path $x \twoheadrightarrow y$ is a total order. Writing n for the number of vertices in G, the function $f : G \to N_I n$, which to a vertex associates the number of vertices strictly below it wrt $\leq$, is universal in the sense of Lemma 4.5. □

Proposition 4.7 *The free conservative finite cocompletion $\mathcal{P}$ of $\mathcal{L}$ is equivalent to the full subcategory of $\hat{\mathcal{L}}$ whose objects are presheaves P satisfying the conditions of Proposition 4.3.*

Proof By Lemma 4.4, the category $\mathcal{P}$ is equivalent to the full subcategory of $\hat{\mathcal{L}}$ whose objects are presheaves preserving limits of diagrams of the form (8) generated by some graph $G \in \hat{\mathcal{G}}$ which admits a colimit, i.e. by Lemma 4.6 the finite graphs which are acyclic and connected. We write G_n for the graph with $[n]$ as vertices and edges $i \to (i+1)$ for $0 \leq i < n-1$. It can be shown that any acyclic and connected finite graph can be obtained from the graph G_n, for some $n \in \mathbb{N}$, by iteratively adding an edge $x \to y$ for some vertices x and y such that there exists a non-empty path $x \twoheadrightarrow y$. Namely, suppose given an acyclic and connected finite graph G. The relation $\leq$ on its vertices, defined by $x \leq y$ whenever there exists a path $x \twoheadrightarrow y$, is a total order, and therefore the graph G contains G_n, where n is the number of edges of G. An edge in G which is not in G_n is necessarily of the form $x \to y$ with $x \leq y$, otherwise it would not be acyclic. Since by Proposition 4.3, see (7), the diagram generated by a graph of the form



is preserved by presheaves in $\mathcal{P}$ (which corresponds to adding an edge between vertices at the source and target of a non-empty path), it is enough to show that presheaves in $\mathcal{P}$ preserve diagrams generated by graphs G_n. This follows again by Proposition 4.3, see (6). □

One can notice that a presheaf $P \in \mathcal{P}$ is characterized by its underlying graph since $P(0)$ is reduced to one element and $P(n)$ with $n > 2$ is the set of paths of length n in this underlying graph: $P \cong I_*(I^* P)$. We can therefore simplify the description of the cocompletion of $\mathcal{L}$ as follows:

Theorem 4.8 *The free conservative finite cocompletion $\mathcal{P}$ of $\mathcal{L}$ is equivalent to the full subcategory of the category $\hat{\mathcal{G}}$ of graphs, whose objects are finite graphs such that for every non-empty path $x \twoheadrightarrow y$ there exists exactly one edge $x \to y$. Equivalently, it can be described as the category whose objects are finite sets equipped with a transitive relation $<$, and functions respecting relations.*

In this category, pushouts can be explicitly described as follows:

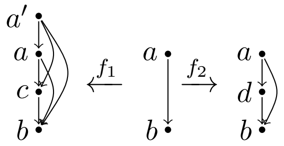
Proposition 4.9 *With the last above description, the pushout of a diagram $(B, <_B) \xleftarrow{f} (A, <_A) \xrightarrow{g} (C, <_C)$ is $B \uplus C/\sim$ with $B \ni b \sim c \in C$ whenever there exists $a \in A$ with $f(a) = b$ and $f(a) = c$, equipped with the transitive closure of the relation inherited by $<_B$ and $<_C$.*

Lines with labels. The construction can be extended to the labeled case (i.e. L is not necessarily a singleton). The forgetful functor $\hat{\mathcal{L}} \to \mathbf{Set}$ sending a presheaf P to the set $P(1)$ admits a right adjoint $! : \mathbf{Set} \to \hat{\mathcal{L}}$. Given $n \in \mathbb{N}^*$ the elements of $!L(n)$ are words u of length n over L, with $!L(s_i^{n-1})(u)$ being the word obtained from u by removing the i-th letter. The free conservative finite cocompletion $\mathcal{P}$ of $\mathcal{L}$ is the slice category $\mathcal{L}/!L$, whose objects are pairs (P, ℓ) consisting of a finite presheaf $P \in \hat{\mathcal{L}}$ together with a *labeling* morphism $\ell : P \to !L$ of presheaves. Alternatively, the description of Proposition 4.8 can be straightforwardly adapted by labeling the elements of the objects by elements of L (labels should be preserved by morphisms), thus justifying the use of labels for the vertices in following examples.

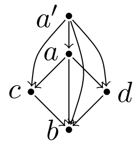
5 Examples

In this section, we give some examples of merging (i.e. pushout) of patches.

Example 5.1 Suppose that starting from a file ab, one user inserts a line a' at the beginning and c in the middle, while another one inserts a line d in the middle. After merging the two patches, the resulting file is the pushout of



which is

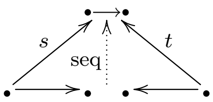


Example 5.2 Write G_1 for the graph with one vertex and no edges, and G_2 for the graph with two vertices and one edge between them. We write $s, t : G_1 \to G_2$

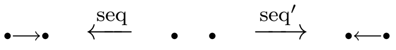
for the two morphisms in $\mathcal{P}$. Since $\mathcal{P}$ is finitely cocomplete, there is a coproduct $G_1 + G_1$ which gives, by universal property, an arrow $\mathrm{seq} : G_1 + G_1 \to G_2$:

$$\begin{tikzcd} & G_2 & \\ G_1 \arrow[r] \arrow[ur, "s"] & G_1 + G_1 \arrow[u, dotted, "\mathrm{seq}"] & G_1 \arrow[l] \arrow[ul, "t"'] \end{tikzcd}$$

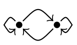
or graphically



that we call the *sequentialization morphism*. This morphism corresponds to the following patch: given two possibilities for a line, a user can decide to turn them into two consecutive lines. We also write $\mathrm{seq}' : G_1 + G_1 \to G_2$ for the morphism obtained similarly by exchanging s and t in the above cocone. Now, the pushout of

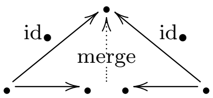


is



which illustrates how cyclic graphs appear in $\mathcal{P}$ during the cocompletion of $\mathcal{L}$.

Example 5.3 With the notations of the previous example, by taking the coproduct of two copies of $\mathrm{id}_{G_1} : G_1 \to G_1$, there is a universal morphism $G_1 + G_1 \to G_1$, which illustrates how two independent lines can be merged by a patch (in order to resolve conflicts).



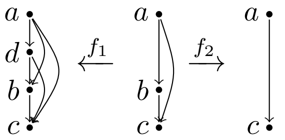
6 Handling deletions of lines

All the steps performed in previous sections in order to compute the free conservative finite cocompletion of the category $\mathcal{L}^+$ can be adapted in order to compute the cocompletion $\mathcal{P}$ of the category $\mathcal{L}$ as introduced in Definition 2.1, thus adding support for deletion of lines in patches. In particular, the generalization of the description given by Theorem 4.8 turns out to be as follows.

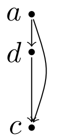
Theorem 6.1 *The free conservative finite cocompletion $\mathcal{P}$ of the category $\mathcal{L}$ is the category whose objects are triples $(A, <, \ell)$ where A is a finite set of lines, $<$ is a transitive relation on A and $\ell : A \to L$ associates a label to each line, and morphisms $f : (A, <_A, \ell_A) \to (B, <_B, \ell_B)$ are partial functions $f : A \to B$ such that for every $a, a' \in A$ both admitting an image under f, we have $\ell_B(f(a)) = \ell_A(a)$, and $a <_A a'$ implies $f(a) <_B f(a')$.*

Similarly, pushouts in this category can be computed as described in Proposition 4.9, generalized in the obvious way to partial functions.

Example 6.2 Suppose that starting from a file abc, one user inserts a line d after a and the other one deletes the line b. The merging of the two patches (in $\mathcal{P}'$) is the pushout of



which is



i.e. the file adc. Notice that the morphism f_2 is partial: b has no image.

Interestingly, a category very similar to the one we have described in Theorem 6.1 was independently proposed by Houston [3] based on a construction performed in [2] for modeling asynchronous processes. This category is not equivalent to ours because morphisms are reversed partial functions: it is thus not the most general model (in the sense of being the free finite cocompletion). As a simplified explanation for this, consider the category **FinSet** which is the finite cocompletion of **1**. This category is finitely complete (in addition to cocomplete), thus $\mathbf{FinSet}^{\mathrm{op}}$ is finitely cocomplete and **1** embeds fully and faithfully in it. However, $\mathbf{FinSet}^{\mathrm{op}}$ is not the finite cocompletion of **1**. Another way to see this is that this category does not contain the "merging" morphism of Example 5.3, but it contains a dual morphism "duplicating" lines.

7 Concluding remarks and future works

In this paper, we have detailed how we could derive from universal constructions a category which suitably models files resulting from conflicting modifications. It is finitely cocomplete, thus the merging of any modifications of the file is well-defined.

We believe that the interest of our methodology lies in the fact that it adapts easily to other more complicated base categories $\mathcal{L}$ than the two investigated here: in future works, we should explain how to extend the model in order to cope with multiple files (which can be moved, deleted, etc.), different file types (containing text, or more structured data such as XML trees). Also, the structure of repositories (partially ordered sets of patches) is naturally modeled by event structures labeled by morphisms in $\mathcal{P}$, which will be detailed in future works, as well as how to model usual operations on repositories: cherry-picking (importing only one patch from another repository), using branches, removing a patch, etc. It would also be interesting to explore axiomatically the addition of inverses for patches, following other works hinted at in the introduction.

Once the theoretical setting is clearly established, we plan to investigate algorithmic issues (in particular, how to efficiently represent and manipulate the conflicting files, which are objects in $\mathcal{P}$). This should eventually serve as a basis for the implementation of a theoretically sound and complete distributed version control system (no unhandled corner-cases as in most current implementations of VCS).

Acknowledgment

The authors would like to thank P.-A. Melliès, E. Haucourt, T. Heindel, T. Hirschowitz and the anonymous reviewers for their enlightening comments and suggestions.

References

[1] J. Adámek and J. Rosicky. *Locally presentable and accessible categories*, volume 189. Cambridge Univ. Press, 1994.

[2] R. Cockett and D. Spooner. Categories for synchrony and asynchrony. *Electronic Notes in Theoretical Computer Science*, 1:66–90, 1995.

[3] R. Houston. *On editing text*. http://bosker.wordpress.com/2012/05/10/on-editing-text.

[4] J. Jacobson. A formalization of darcs patch theory using inverse semigroups. Technical report, CAM report 09-83, UCLA, 2009.

[5] M. Kelly. *Basic concepts of enriched category theory*, volume 64. Cambridge Univ. Press, 1982.

[6] S. Mac Lane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer Verlag, 1971.

[7] S. Mac Lane and I. Moerdijk. *Sheaves in geometry and logic: A first introduction to topos theory*. Springer, 1992.

[8] R. Paré. Connected components and colimits. *Journal of Pure and Applied Algebra*, 3(1):21–42, 1973.

[9] M. Ressel, D. Nitsche-Ruhland, and R. Gunzenhäuser. An integrating, transformation-oriented approach to concurrency control and undo in group editors. In *Proceedings of the 1996 ACM conference on Computer supported cooperative work*, pages 288–297. ACM, 1996.

[10] D. Roundy and al. *The Darcs Theory*. http://darcs.net/Theory.

[11] R. Street and R. Walters. The comprehensive factorization of a functor. *Bull. Amer. Math. Soc*, 79(2):936–941, 1973.

Monads for Behaviour

Maciej Piróg[1] and Jeremy Gibbons[2]

Department of Computer Science
University of Oxford

Abstract

The monads used to model effectful computations traditionally concentrate on the 'destination'—the final results of the program. However, sometimes we are also interested in the 'journey'—the intermediate course of a computation—especially when reasoning about non-terminating interactive systems. In this article we claim that a necessary property of a monad for it to be able to describe the behaviour of a program is complete iterativity. We show how an ordinary monad can be modified to disclose more about its internal computational behaviour, by applying an associated transformer to a completely iterative monad. To illustrate this, we introduce two new constructions: a coinductive cousin of Cenciarelli and Moggi's generalised resumption transformer, and States—a State-like monad that accumulates the intermediate states.

Keywords: completely iterative monads, effects, tracing, resumptions

1 Introduction

In this article we are concerned with semantics of programs like the following Haskell fragment:

```
echo :: IO ()
echo = do { x <- getChar ; putChar x ; echo }
```

More precisely, we are interested in programs that (1) have side-effects, and (2) depend on a (not necessarily terminating) recursion—or a corecursion, if you will. In the example, `echo` performs observable actions and then calls itself, 'unfolding' an infinite series of events.

Since Moggi's groundbreaking work [20], monads have become the standard model for computational effects. A popular choice for I/O is to employ the State monad $A \mapsto (A \times S)^S$, model the outside world as an object S, and see the program semantics as a function transforming an initial state into a final state [7,15]. Alternatively, we could consider side-effects as communication with the environment, so

[1] `maciej.pirog@cs.ox.ac.uk`

[2] `jeremy.gibbons@cs.ox.ac.uk`

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: `www.elsevier.com/locate/entcs`

no assumption about semantics of effects needs to be made at this point: the program semantics is a free structure generated by the 'effectful' constructs (`getChar` and `putChar`), which is then interpreted by an external handler [13,25,28].

The situation becomes much more complicated in the context of (2). For example, the State monad does not build the final state incrementally, so in case of non-terminating programs, such as `echo`, it is useless. The free structure, on the other hand, sometimes needs to be infinite, so in general the free monad Σ^* (for an endofunctor Σ representing the signature) is 'too small'. Evidently, to encompass these examples we need monads that capture not only the final results of the program, but rather its behaviour, for example in the form of execution traces. In this article we identify this property as *complete iterativity*, a notion introduced by Elgot [10] and recently brought to attention by Aczel *et al.* [1,18]. A monad is completely iterative ('is a cim') if it is equipped with a certain corecursion scheme that is coherent with its monadic structure (for the full definition see Section 2). In particular, the free cim Σ^∞ generated by an endofunctor Σ captures both finite and infinite Σ-terms.

Nevertheless, we should not discard the 'usual' monads too hastily. For example, if we program a divergent computation in the State monad, the intermediate states are physically 'put' and 'gotten' somewhere in the memory of the computer, so the internal behaviour of the computation is, in a sense, accurate. The point is to reify it as a mathematical model. An interesting fact is that the `IO` monad in the Haskell Glasgow Compiler (GHC) is implemented using the State monad [17], so whatever its mathematical model, the two have to be related.

Our idea is to use transformers associated with the 'usual' monads to trace computations. For a cim T and an adjunction $F \dashv U$ that gives rise to a monad M (that is, $UF = M$), we use the monad UTF to trace computations in M. Clearly, UTF supports M-computations (via the canonical monad morphism $M \to UTF$), but it can also store some observations about the course of the computation in the inner cim. The choice of the monad T and the adjunction reveals different aspects of computations in M. As our main technical result, we prove that UTF is completely iterative.

As an example, we use the currying adjunction to derive what we call the States monad, which behaves like State, but also gathers the intermediate states in a stream. This way, the result of the computation is not a single, final state, but rather a possibly infinite trace consisting of intermediate states.

Then we introduce the Coinductive Generalised Resumption transformer $M(\Sigma M)^\infty$, which is a coalgebraic cousin of Cenciarelli and Moggi's Generalised Resumption transformer $M(\Sigma M)^*$ [9]. It allows to decompose a monadic computation into a possibly infinite number of steps interleaved with free structure. It is also a categorical model for datatypes built around resumptions, such as Haskell pipes (for $\Sigma A = A^I + A \times O$). The fact that we use the free cim is crucial, since programming patterns for pipes rely heavily on infinite computations.

Because of space limitations we present only short outlines of proofs. For the full proofs consult the associated technical report available online at: `http://www.cs.ox.ac.uk/people/maciej.pirog/mbext.pdf`.

2 Completely iterative monads

2.1 Initial assumptions and notations

For the entire article, we assume that we are working in a base category $\mathcal{B}$ with binary coproducts and all the necessary final coalgebras. We denote the coproduct injections by inl and inr. We use a subscript for the composition of a natural transformation with a functor; for example, for functors H and J, if $\xi : F \to G$ is natural, then $\xi_H : FH \to GH$. If ξ is natural in two variables, by $\xi_{H,J}$ we mean a natural transformation $\zeta_A = \xi_{HA,JA}$.

Working with infinite computations means working also with infinite data structures. To set the notation, we recall a few standard definitions. For an endofunctor F, an F-coalgebra is a pair $\langle A, f : A \to FA \rangle$. We call A the *carrier* of the coalgebra. A morphism $h : A \to B$ is an F-coalgebra *homomorphism*, denoted as $h : \langle A, f \rangle \to \langle B, g \rangle$, if $g \cdot h = Fh \cdot f$. An F-coalgebra $\langle \nu F, \beta \rangle$ is *final* if for every F-coalgebra $\langle A, f \rangle$ there exists precisely one homomorphism $\langle A, f \rangle \to \langle \nu F, \beta \rangle$, called an *anamorphism* and denoted as $[\![f]\!]$.

2.2 Ideal and idealised monads

In this article we deal with monads that support corecursion: infinite computations are described by single steps. However, a step might not produce any observable behaviour, for example if it is a pure computation constructed with the unit, or we want to be more selective about which monadic actions are observable. To formalise productive computations, we need the notion of ideals of monads. These are analogous to ideals in a ring or a semigroup—subsets closed under the operations. (All the definitions in this section are as given by Adámek, Milius, and Velebil [2].)

Definition 2.1 *Let $\langle M, \eta, \mu \rangle$ be a monad. For an endofunctor $\overline{M}$, a natural transformation $\sigma : \overline{M} \to M$ with monomorphic components is called a subfunctor of M. We call σ an ideal of M if there exists a natural transformation $\overline{\mu} : \overline{M}M \to \overline{M}$ such that the following diagram commutes.*

$$\begin{tikzcd}
\overline{M}M \arrow[r, "\sigma_M"] \arrow[d, "\overline{\mu}"] & M^2 \arrow[d, "\mu"] \\
\overline{M} \arrow[r, "\sigma"] & M
\end{tikzcd}$$

We call a pair of a monad and its ideal an idealised monad. An idealised monad M is called an ideal monad if $M = \mathsf{Id} + \overline{M}$ with $\eta = \mathsf{inl}_{\mathsf{Id},\overline{M}}$ and $\sigma = \mathsf{inr}_{\mathsf{Id},\overline{M}}$.

Examples of ideal monads include: free monads, exceptions, interactive output, and nonempty lists. Note that in a category with an initial object 0, every monad M is idealised with respect to the trivial ideal $FX = 0$, that is a constant functor that always returns the initial object.

We also need morphisms that respect the internal structure of idealised monads. If Σ is an endofunctor, then a natural transformation $\xi : \Sigma \to M$ is ideal if its codomain contains only productive computations. Intuitively, this means that an interpretation of a symbol from the signature should never yield a pure computation. Formally:

Definition 2.2 *Let $\langle M, \sigma^M \rangle$ and $\langle N, \sigma^N \rangle$ be idealised monads. A natural transformation $\xi : \Sigma \to M$ is ideal if it factors through σ^M.*

2.3 Cims defined

For an idealised monad M, we describe a step of a computation by a morphism of type $e : X \to M(A + X)$, called an *equation morphism.* The object X represents (a set of) *variables*—the seeds of the corecursion. The object A represents (a set of) *parameters*, which are final values of the computation. An equation morphism is productive (is *guarded*) if it always produces effects (in the sense of idealised monads) or a final value, but not a variable:

Definition 2.3 *A morphism $e : X \to M(A + Y)$ is guarded if it factors through the morphism $[\sigma_{A+Y}, \eta_{A+Y} \cdot \mathsf{inl}_{A,Y}]$, that is there exists a morphism j such that the following diagram commutes.*

$$\begin{tikzcd}
X \arrow[r, "e"] \arrow[d, dashed, "j"] & M(A+Y) \\
\overline{M}(A+Y)+A \arrow[ur, "{[\sigma_{A+Y},\ \eta_{A+Y} \cdot \mathsf{inl}_{A,Y}]}"'] &
\end{tikzcd}$$

If $X = Y$, we call e a guarded equation morphism.

We use a guarded equation morphism e to unfold a computation $e^\dagger$, called a solution. Intuitively, a solution is an infinite iteration of parameter-preserving Kleisli-compositions of e. A monad is a cim if such a composition always exists and is unique. Formally:

Definition 2.4 *Let $e : X \to M(A + X)$ be a morphism. We call a morphism $e^\dagger : X \to MA$ a solution of e if the following diagram commutes.*

$$\begin{tikzcd}
X \arrow[r, "e^\dagger"] \arrow[d, "e"] & MA \\
M(A+X) \arrow[r, "{M[\eta_A, e^\dagger]}"] & M^2A \arrow[u, "\mu_A"']
\end{tikzcd}$$

An idealised monad M is completely iterative if every guarded equation morphism has a unique solution.

Cims make it possible to separate the corecursion guarded by invocation of effects from a recursive structure of the base category, like order or metric enrichment. This separation is important conceptually. Consider a server dealing with some requests: though it is non-terminating, it probably does not require unbounded recursion in between handling two requests.

Conversely, in a language with unbounded recursion, M-computations consisting of guarded steps are necessarily solutions: An infinite computation can be seen as the colimit of the ω-chain consisting of single steps. Consider an ω-chain $\{f_i : X_i \to MX_{i+1}\}_{i\in\mathbb{N}}$ of Kleisli morphisms that factor through $\sigma^M_{X_{i+1}}$. In a category with countable coproducts, we define a guarded equation morphism $e = [(id_0 + M\mathsf{in}_{i+1}) \cdot f_i]_{i\in\mathbb{N}} : \coprod_{i\in\mathbb{N}} X_i \to M(0 + \coprod_{i\in\mathbb{N}} X_i)$. One can show that the family of morphisms $\{e^\dagger \cdot \mathsf{in}_i : X_i \to M0\}_{i\in\mathbb{N}}$ is the colimit of the chain in the Kleisli category of M.

2.4 The free cim

An example of a cim is a generalisation of the infinite term monad generated by an endofunctor (intuitively, a signature) Σ. Its functorial part is given by a family of final coalgebras $\Sigma^\infty A = \nu X.A + \Sigma X$. Below we define the unit, η^∞, and a natural transformation $\mathsf{emb} : \Sigma \to \Sigma^\infty$ that embeds Σ in Σ^∞. For an explicit definition of the multiplication μ^∞ refer to Section 5, and put Id for M in the definition of μ^K.

$$\begin{tikzcd}
\mathsf{Id} \arrow[d, "\eta^\infty = \mathsf{inl}_{\mathsf{Id},\Sigma\Sigma^\infty}"] \\
\mathsf{Id} + \Sigma\Sigma^\infty \cong \Sigma^\infty
\end{tikzcd}
\qquad
\begin{tikzcd}
\Sigma \arrow[d, "\mathsf{emb} = \mathsf{inr}_{\mathsf{Id},\Sigma\Sigma^\infty} \cdot \Sigma\eta^\infty"] \\
\mathsf{Id} + \Sigma\Sigma^\infty \cong \Sigma^\infty
\end{tikzcd}$$

As discussed by Aczel *et al.* [1], Σ^∞ is the free cim generated by Σ. Intuitively, this means that every interpretation of Σ in a cim M extends in a unique way to an interpretation of the entire (possibly infinite) term Σ^∞ in M. Formally, for an ideal natural transformation $\xi : \Sigma \to M$, there exists a unique monad morphism $\iota(\xi) : \Sigma^\infty \to M$ such that the following diagram commutes.

$$\begin{tikzcd}
\Sigma \arrow[r, "\mathsf{emb}"] \arrow[dr, "\xi"'] & \Sigma^\infty \arrow[d, "\iota(\xi)"] \\
& M
\end{tikzcd}$$

The monad morphism $\iota(\xi)$ is given by $[\eta^M, \xi^\dagger_{\Sigma^\infty}]$. Diagrammatically:

$$\begin{tikzcd}
\Sigma\Sigma^\infty \arrow[d, "\xi_{\Sigma^\infty}"] \\
M\Sigma^\infty \cong M(\mathsf{Id} + \Sigma\Sigma^\infty)
\end{tikzcd}
\qquad
\begin{tikzcd}
\Sigma\Sigma^\infty \arrow[d, "\xi^\dagger_{\Sigma^\infty}"] \\
M\mathsf{Id} = M
\end{tikzcd}
\qquad
\begin{tikzcd}
\Sigma^\infty \cong \mathsf{Id} + \Sigma\Sigma^\infty \arrow[d, "{\iota(\xi) = [\eta^M, \xi^\dagger_{\Sigma^\infty}]}"] \\
M
\end{tikzcd}$$

Another example of a cim is the Exception monad $A \mapsto A + E$. Also, every monad is completely iterative with respect to the trivial ideal $FX = 0$. But, except for those and the free cim, there are hardly any examples of cims commonly used in programming or semantics. This paper aims to fill this void in a rather generic fashion.

3 Cims, adjunctions, and tracing

Let M be a monad, and let $\langle F, U, \eta, \varepsilon\rangle : \mathcal{B} \to \mathcal{C}$ be a factorization of M as an adjunction, that is $M = \langle UF, \eta, U\varepsilon_F\rangle$. Let $\langle T, \eta^T, \mu^T, \sigma^T\rangle$ be a cim with solutions $\text{-}^\dagger$. It is standard that UTF is a monad with $\eta^{UTF} = U\eta^T_F \cdot \eta$ and $\mu^{UTF} = U\mu^T_F \cdot UT\varepsilon_{TF}$, and that $\mathsf{lift} = U\eta^T_F : UF \to UTF$ is a monad morphism. We prove that UTF inherits complete iterativity from T.

Theorem 3.1 *The natural transformation $U\sigma^T_F : U\overline{T}F \to UTF$ forms an ideal. The monad UTF is completely iterative with respect to this ideal.*

Proof. Right adjoints preserve monomorphisms, hence the components of the natural transformation $U\sigma^T_F$ are monic, and so it is a subfunctor. We define $\overline{\mu}$ to be $U\overline{\mu^T_F} \cdot U\overline{T}\varepsilon_{TF}$. It is easy to verify that it satisfies the condition for ideals.

Let $e : X \to UTF(A + X)$ be a $U\sigma^T_F$-guarded equation morphism. By $\lfloor\text{-}\rfloor : \mathcal{C}[FA, B] \cong \mathcal{B}[A, UB] : \lceil\text{-}\rceil$ we denote the natural isomorphism associated with the adjunction. Recall that left adjoints preserve coproducts, that is $F(A+B) \cong FA + FB$. One can calculate that $\lceil e\rceil \cong [\sigma^T_{(FA+FX)}, \eta^T_{(FA+FX)} \cdot \mathsf{inl}_{(FA,FX)}] \cdot (\varepsilon_{\overline{T}F(A+X)} + \mathsf{id}_{FA}) \cdot Fj$, which means that $\lceil e\rceil : FX \to TF(A + X) \cong T(FA + FX)$ is a guarded equation morphism in T with a unique solution $\lceil e\rceil^\dagger : FX \to TFA$.

We define the solution of e as $\lfloor\lceil e\rceil^\dagger\rfloor$. The following diagram commutes:

$$\begin{tikzcd}
X \arrow[rrr, bend left, "\lfloor\lceil e\rceil^\dagger\rfloor = U\lceil e\rceil^\dagger \cdot \eta_X"] \arrow[dr, "\eta_X"] \arrow[ddr, bend right, "e"'] & & & \\
& UFX \arrow[rr, "U\lceil e\rceil^\dagger"] \arrow[d, "U\lceil e\rceil"] & & UTFA \\
& \begin{array}{c} UTF(A+X) \\ \cong UT(FA+FX) \end{array} \arrow[rr, "UT[\eta^T_{FA}, \lceil e\rceil^\dagger]"] \arrow[drrr, bend right, "UTF[\eta^{UTF}_A, U\lceil e\rceil^\dagger \cdot \eta_X]"'] & & UT^2FA \arrow[u, "U\mu^T_{FA}"] & \\
& & & & (UTF)^2A \arrow[ul, "UT\varepsilon_{TFA}"] \arrow[uul, bend right, "\mu^{UTF}_A"']
\end{tikzcd}$$

The inner square is the U-image of the solution diagram for $\lceil e\rceil^\dagger$. The outer triangles commute due to properties of adjunctions and the definition of μ^{UTF}.

For uniqueness, let $g : X \to UTFA$ be a solution of e. Substitute $\lceil g\rceil$ for $\lceil e\rceil^\dagger$ in the above diagram. The outer square commutes, because $\lfloor\lceil g\rceil\rfloor = g$ is a solution, and the triangles commute, because of properties of adjunctions, hence the inner square precomposed with η_X also commutes. For all morphisms $f, f' : FB \to C$, if $Uf \cdot \eta_B = Uf' \cdot \eta_B$ then $f = f'$. Therefore, $\lceil g\rceil$ is a solution of $\lceil e\rceil$, so $\lceil g\rceil = \lceil e\rceil^\dagger$, hence $g = \lfloor\lceil g\rceil\rfloor = \lfloor\lceil e\rceil^\dagger\rfloor$. □

Intuitively, T collects observations about a computation in M. Thus, we need a new operation that allows us to actually observe the current state of the computation, for example the current state in the State monad (this example is elaborated in the next section). It could be given as a natural transformation $\mathsf{olift} : M \to UTF$ with components that factor through $U\sigma^T_F$. It will not in general be a monad morphism; on the contrary, performing two actions and then observing the effect differs in general from observing the effect of each action individually. More formally, let $f \circ g$ be a computation in the Kleisli category of M, where $\circ$ is the Kleisli composition. We can decorate it with observers in two different ways: $\mathsf{olift} \cdot (f \circ g)$ or $(\mathsf{olift} \cdot f) \circ (\mathsf{olift} \cdot g)$. For example, when tracing a computation in State, we may want to observe only 'put' operations, as long as we are certain that there are only finitely many invocations of 'get' in between every two invocations of 'put'. In the rest of the paper we always define olift as $U\mathsf{obs}$ for a natural transformation $\mathsf{obs} : F \to TF$. For convenience, we also define a 'save the current state of computation' operation $\mathsf{save} = \mathsf{olift} \cdot \eta : \mathsf{Id} \to UTF$.

Though we do not use this property directly in the rest of the article, observations should not modify the computation. This could be captured by the following cancellation property: for all morphisms $f, f' : A \to MB$ and $g, g' : B \to MC$, if $(\mathsf{lift} \cdot g) \circ \mathsf{save}_B \circ (\mathsf{lift} \cdot f) = (\mathsf{lift} \cdot g') \circ \mathsf{save}_B \circ (\mathsf{lift} \cdot f')$ then $g \circ f = g' \circ f'$.

4 The States monad

Our first example is a monad we call States. If the base category $\mathcal{B}$ is cartesian closed, the State monad arises from the currying adjunction $- \times S \dashv -^S$. We choose $(- \times S)^\infty$, for which we write $\overrightarrow{S}$, to be the inner cim, and the result is the monad $A \mapsto (\overrightarrow{S}(A \times S))^S$. Intuitively, $\overrightarrow{S}$ is a possibly infinite stream of states of type S. The 'base' of the exponential is the trace of the computation: a stream that, if finite, is terminated with an answer A and a current state S. The latter is used only to compose two computations and is not stored in the stream.

We define 'put' and 'get' operations as standard liftings of 'put' and 'get' for State. The natural transformation obs duplicates the current state and puts it in the stream as follows, where $\mathsf{outl}_{A,B} : A \times B \to A$ and $\mathsf{outr}_{A,B} : A \times B \to B$ are the left and right projections respectively.

$$\begin{tikzcd}
A \times S \arrow[r, "\langle\langle \mathsf{outl}, \mathsf{outr}\rangle, \mathsf{outr}\rangle"] & (A \times S) \times S \arrow[r, "\mathsf{emb}_{A\times S}"] & \overrightarrow{S}(A \times S)
\end{tikzcd}$$

For example, consider the following computation in States on **Set** for $S = \mathbb{N}$ (using Haskell syntax):

```
let f = do {put 2; save; put 3; save; put 5}
    g = do {x <- get; put (x+1); save; g}
in do {f; g}
```

For any initial state, `f` evaluates to the trace $(2, 3, \langle\star, 5\rangle)$, while the whole computation evaluates to $(2, 3, 6, 7, 8, 9, \ldots)$.

4.1 Example: Control structures for While

Consider a generalised While language, as given by Rutten [26]:

$$P, Q \;::=\; A \mid P; Q \mid \textbf{if } b \textbf{ then } P \textbf{ else } Q \mid \textbf{while } b \textbf{ do } P$$

For a monad M, the symbol A represents a set of actions (denoted as $\underline{a}$), that is morphisms of type $1 \to M1$. The symbol b represents elements of a set B of Boolean expressions, that is a set of morphisms of type $1 \to M(1+1)$. We parametrise the semantics with a 'guard' operation $\gamma : 1 \to M1$, which allows the addition of behaviour on every choice point of a control structure. The denotation of a program P is given by $\llbracket P \rrbracket : 1 \to M1$, defined as follows, where $\circ$ is Kleisli composition.

$$\begin{aligned}
\llbracket \underline{a} \rrbracket &= \underline{a} \\
\llbracket P; Q \rrbracket &= \llbracket Q \rrbracket \circ \llbracket P \rrbracket \\
\llbracket \textbf{if } b \textbf{ then } P \textbf{ else } Q \rrbracket &= [\llbracket P \rrbracket, \llbracket Q \rrbracket] \circ b \circ \gamma \\
\llbracket \textbf{while } b \textbf{ do } P \rrbracket &= ([M\mathsf{inr}_{1,1} \cdot \llbracket P \rrbracket,\; M\mathsf{inl}_{1,1} \cdot \eta^M_1] \circ b \circ \gamma)^\dagger
\end{aligned}$$

Actions denote themselves, and compositions of programs are just Kleisli compositions of morphisms. The denotation of **if** statements first performs the guard γ, then b, and then the appropriate branch is chosen (we use the left component of $1+1$ to represent 'true'). The denotation of **while** first builds an equation morphism by composing the guard, the condition, and the choice between returning the left component of the coproduct (a constant, which means 'stop the iteration'), or performing the body, and right-injecting the result (which makes it a 'continue the iteration' variable). The denotation of the entire **while** expression is a solution to that morphism. The solution might not exist, or might not be unique; hence, depending on the choice of M, A, B, and γ, the denotation might not be well-defined. This semantics specialises to a couple of known cases:

If we choose the regular State monad on **Dcppo** (the category of pointed directed-complete partial orders and continuous functions) for M and its unit on 1 for γ, the solution diagram simplifies to the familiar equation for denotation of While [23, Chapter 4]. So, if we assume $\text{-}^\dagger$ to be the least fixed point, we yield the standard denotational semantics.

If we instantiate M with a cim, we can ensure that unique solutions always exist by an appropriate γ-guarding of **while** loops. (Note that it is not sufficient to ask for the A actions to be guarded, since **while true do while false do** $\underline{a}$ diverges without invoking an action.) In case of the States monad, this means that every iteration stores its initial state in the stream, that is $\gamma = \mathsf{save}$. Additionally, if we assume that 'put' operations are always guarded and 'get' are not, we obtain a semantics trace-equivalent to Nakata and Uustalu's trace operational semantics [22].

5 Coinductive generalised resumptions

Let $\langle M, \eta^M, \mu^M \rangle$ be a monad, and Σ be an endofunctor on the base category $\mathcal{B}$. In this section we give a monadic structure to $M(\Sigma M)^\infty$ and examine its basic properties. We proceed by first giving a monadic structure to the endofunctor

$$KA = \nu X.M(A + \Sigma X),$$

which is isomorphic to $M(\Sigma M)^\infty$ through the coalgebraic version of the rolling rule [5]:

Lemma 5.1 *Let F, G be endofunctors. Then $\nu FG \cong F\nu GF$.*

For convenience, we define two auxiliary natural transformations. The first one, $\mathsf{flat}_{A,B} : M(MA+B) \to M(A+B)$, flattens a computation that may return a value or a new computation. The second one, $\mathsf{unf} : K^2 \to M(\mathsf{Id} + \Sigma K^2)$, unfolds and flattens two levels of structure of K. Note that the final coalgebra map $\alpha_A : KA \to M(A + \Sigma KA)$ is natural in A.

$$\begin{tikzcd}
\mathsf{flat}_{A,B} = & M(MA+B) \arrow[d, "M(\mathsf{id}_{MA} + \eta^M_B)"] \\
& M(MA+MB) \arrow[d, "M[M\mathsf{inl}_{A,B}, M\mathsf{inr}_{A,B}]"] \\
& M^2(A+B) \arrow[d, "\mu^M_{A+B}"] \\
& M(A+B)
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathsf{unf} = & K^2 \arrow[d, "\alpha_K"] \\
& M(K + \Sigma K^2) \arrow[d, "M(\alpha + \mathsf{id}_{\Sigma K^2})"] \\
& M(M(\mathsf{Id} + \Sigma K) + \Sigma K^2) \arrow[d, "\mathsf{flat}_{\mathsf{Id}+\Sigma K, \Sigma K^2}"] \\
& M(\mathsf{Id} + \Sigma K + \Sigma K^2)
\end{tikzcd}$$

The unit (return) η^K of the monad K is given below. The multiplication (join) is defined as the anamorphism $\mu^K_A = [\!(m_A)\!]$ for the following natural transformation m.

$$\begin{tikzcd}
\eta^K = & \mathsf{Id} \arrow[d, "\mathsf{inl}_{\mathsf{Id},\Sigma K}"] \\
& \mathsf{Id} + \Sigma K \arrow[d, "\eta^M_{\mathsf{Id}+\Sigma K}"] \\
& M(\mathsf{Id} + \Sigma K) \cong K
\end{tikzcd}
\qquad
\begin{tikzcd}
m = & K^2 \arrow[d, "\mathsf{unf}"] \\
& M(\mathsf{Id} + \Sigma K + \Sigma K^2) \arrow[d, "M(\mathsf{id} + {[\Sigma \eta^K_K, \mathsf{id}_{\Sigma K^2}]})"] \\
& M(\mathsf{Id} + \Sigma K^2)
\end{tikzcd}$$

Theorem 5.2 *The following hold:*

(i) *The tuple $\langle K, \eta^K, \mu^K \rangle$ is a monad,*

(ii) *It is compatible [6, Chapter 9] with M and $(\Sigma M)^\infty$, which yields a monad distributive law $\lambda : (\Sigma M)^\infty M \to M(\Sigma M)^\infty$,*

(iii) *There exist two monad morphisms $\mathsf{liftl} : M \to M(\Sigma M)^\infty$ and $\mathsf{liftr} : \Sigma^\infty \to M(\Sigma M)^\infty$.*

Proof outline. The statements (i) and (ii) can be proved by the structural coinduction provided by the finality of K. The distributive law induces two canonical monad morphisms $M \to K$ and $(\Sigma M)^\infty \to K$. We compose the latter with a monad morphism $\Sigma^\infty \to (\Sigma M)^\infty$ given by $\iota(\mathsf{emb} \cdot \Sigma\eta^M)$. □

Despite the existence of the cospan $M \to M(\Sigma M)^\infty \leftarrow \Sigma^\infty$, the monad $M(\Sigma M)^\infty$ is in general not a coproduct of M and Σ^∞ as monads. To see that, it is sufficient to assume that the base category is **Set**, M is ideal, and to recall the construction of coproducts of ideal monads by Ghani and Uustalu [12]. In such a setting the coproduct allows only a finite number of interleavings between M and Σ^∞, so it is distinct from K.

5.1 Complete iterativity of K

Consider the subcategory M-**Fema** of free Eilenberg-Moore M-algebras (that is, algebras where the carrier is of the shape MA, and the action is defined as μ^M_A). It is equivalent to the Kleisli category for M. There is a standard free-underlying adjunction $F \dashv U : \mathcal{B} \to M$-**Fema**.

As discussed by Mulry [21], liftings of an endofunctor T on $\mathcal{B}$ to M-**Fema** are in one-to-one correspondence with distributive laws $TM \to MT$. Moreover, a simple calculation shows that if T has a monadic structure and the distributive law respects this structure, the corresponding lifting $\langle T\rangle$ is also a monad. The monad MT induced by the distributive law is equal to the monad $U\langle T\rangle F$.

Consider the monad $(\Sigma M)^\infty$. The monad distributive law λ from Theorem 5.2 gives rise to a lifting $\langle(\Sigma M)^\infty\rangle$, defined on objects as $\langle(\Sigma M)^\infty\rangle MA = M(\Sigma M)^\infty A \cong KA$. The following theorem states that the lifting is also a free cim (note that $M\Sigma$ is an endofunctor also over M-**Fema**):

Theorem 5.3 *The monad $\langle(\Sigma M)^\infty\rangle$ is the free cim generated by $M\Sigma$ in M-**Fema**.*

Proof outline. For an M-**Fema** morphism $f : MX \to M(A + \Sigma MX)$ the finality diagram for KA commutes also in M-**Fema**. The definition of coproducts $\oplus$ in M-**Fema** yields $M(A + \Sigma -) = MA \oplus M\Sigma -$, which makes the finality diagram a finality diagram for $MA \oplus M\Sigma -$. This means that $KA \cong \langle(\Sigma M)^\infty\rangle MA$ is the carrier of the final $(MA \oplus M\Sigma -)$-coalgebra, and so, according to [18, Corollary 6.3], $\langle(\Sigma M)^\infty\rangle$ is the functorial part of the free cim generated by $M\Sigma$ understood as a functor in M-**Fema**. The fact that the monadic structures of the lifting and the free cim in M-**Fema** are equal can be proved by a simple coinduction. □

Theorem 3.1 and the above characterisation yield that K is completely iterative. The guardedness condition specialises as:

$$\begin{tikzcd}
X \arrow[r, "e"] \arrow[d, dashed, "j"] & K(A+X) \\
M\Sigma K(A+X) + A \arrow[ur, "{[\alpha^{-1}_{A+X} \cdot M\mathsf{inr}_{A+X,\Sigma K(A+X)},\ \eta^K_{A+X} \cdot \mathsf{inl}_{A,X}]}"'] &
\end{tikzcd}$$

5.2 Example: Bisimulation

Let $\Sigma = \mathsf{Id}$, so that $K \cong MM^\infty$. Similarly to Cenciarelli and Moggi's transformer MM^* [9], a K-computation can be seen as an M-computation split into a series of suspended steps. However, in case of MM^∞, the structure can be infinite, so it can also store a divergent computation. We can see the result of each step as a rather robust observation about the current state of the computation. So, even if the computation does not have a final value, we can still reason about the course of the computation.

We define the natural transformation $\mathsf{obs} : M \to MM^\infty$ as:

$$\begin{tikzcd}
M \arrow[r, "M\eta^M"] & MM \arrow[r, "M\mathsf{emb}"] & MM^\infty
\end{tikzcd}$$

It builds an empty level, so that a composition with another value will not affect the current structure. Intuitively, the outer M is the current state of the computation, while M^∞ is a kind of continuation. To acquire the second state, we can contract the top two steps of execution using a natural transformation force defined as follows, where flat' is equal to flat, but with the monadic argument as the second component of the coproduct rather than the first.

$$\begin{tikzcd}
MM^\infty \cong & M(\mathsf{Id} + MM^\infty) \arrow[d, "\mathsf{flat}'_{\mathsf{Id},M^\infty}"] \\
M(\mathsf{Id} + M^\infty) \cong & M(\mathsf{Id} + \mathsf{Id} + MM^\infty) \arrow[d, "M({[\mathsf{id},\mathsf{id}]} + \mathsf{id}_{MM^\infty})"] \\
& M(\mathsf{Id} + MM^\infty) \cong MM^\infty
\end{tikzcd}$$

On **Set**, we can define a simple notion of *bisimulation* between programs as a predicate $\approx\, \subseteq (MM^\infty A)^2$, such that for $p, q \in MM^\infty A$, it is the case that $p \approx q$ precisely if $M(\mathsf{id}_A + !_{M^\infty A})(p) = M(\mathsf{id}_A + !_{M^\infty A})(q)$ and $\mathsf{force}(p) \approx \mathsf{force}(q)$, where $!_A : A \to 1$ is the unique morphism to the final object. In other words, we compare the functorial structure of the outer M (the observable result of the first step), and continue the process after performing the next step with the force natural transformation. This means that two programs are bisimilar if for every $n \in \mathbb{N}$, the respective prefixes of performing the first n steps are equal.

6 Related and future work

Cims arise from completely iterative algebras. Both concepts have been extensively studied by Elgot [10] and by Aczel *et al.* [1,18]. Milius and Moss [19] consider recursive program schemes in terms of solutions in Elgot algebras [3] (that is, Eilenberg-Moore algebras for free cims).

Cenciarelli and Moggi [9] introduced the Generalised Resumption transformer $M(\Sigma M)^*$, which decomposes a monadic computation into a series of steps (layers of free structure). Hyland, Plotkin, and Power [16] proved it to be the coproduct $M + \Sigma^*$ in the category of monads. The monad $M(\Sigma M)^\infty$ captures also potentially infinite computations. In some categories—and so programming languages like Haskell—the limit-colimit coincidence [27] identifies $M(\Sigma M)^*$ and $M(\Sigma M)^\infty$, but the explicit use of the free cim is significant in **Set** and in type theories with guarded (co)recursion. Interleaving data and monadic actions is a powerful abstraction studied recently also by Filinski and Støvring [11], Atkey *et al.* [4], and the present authors [24].

Since the free cim is a final coalgebra [18], we can see $(M\Sigma)^\infty$ in M-**Fema** from Theorem 5.3 as an example of Hasuo, Jacobs, and Sokolova's generic trace semantics [14], which models state-based systems as F-coalgebras in a Kleisli category (or, equivalently, a **Fema**). The coalgebra represents transitions (for example, with $\Sigma A = A \times O$ for labelled transitions), and the monad represents the underlying effect (like the Powerset monad for nondeterminism or the Probability Distribution monad for probabilistic systems).

In this paper we concentrate on the monads and tracing, and we only sketch potential applications in defining semantics and reasoning about programs. The natural next step is to formalise a language like Moggi's computational λ-calculus [20] with recursion provided by a background cim. It is also an interesting question whether the presented theory could be used to develop a practical framework for reasoning about effectful programs in type theories, like those implemented by the Coq or Agda proof systems. So far, Capretta [8] represented general recursion by the free cim generated by the identity functor; we conjecture fruitful applications of other cims too.

Acknowledgments

This work was supported by the UK EPSRC project *Reusability and Dependent Types* (EP/G034516/1). We would like to thank Marek Materzok for his useful notes and the anonymous reviewers for the detailed and encouraging comments.

References

[1] Peter Aczel, Jirí Adámek, Stefan Milius, and Jiri Velebil. Infinite trees and completely iterative theories: a coalgebraic view. *Theoretical Computer Science*, 300(1-3):1–45, 2003.

[2] Jirí Adámek, Stefan Milius, and Jiri Velebil. On rational monads and free iterative theories. *Electronic Notes in Theoretical Computer Science*, 69:23–46, 2002.

[3] Jirí Adámek, Stefan Milius, and Jiri Velebil. Elgot algebras. *Logical Methods in Computer Science*, 2(5), 2006.

[4] Robert Atkey, Neil Ghani, Bart Jacobs, and Patricia Johann. Fibrational induction meets effects. In Lars Birkedal, editor, *Foundations of Software Science and Computational Structures—15th International Conference, FoSSaCS 2012*, volume 7213 of *Lecture Notes in Computer Science*, pages 42–57. Springer, 2012.

[5] Roland Carl Backhouse, Marcel Bijsterveld, Rik van Geldrop, and Jaap van der Woude. Categorical fixed point calculus. In David H. Pitt, David E. Rydeheard, and Peter Johnstone, editors, *Category Theory and Computer Science*, volume 953 of *Lecture Notes in Computer Science*, pages 159–179. Springer, 1995.

[6] Michael Barr and Charles F. Wells. *Toposes, Triples, and Theories.* Grundlehren der mathematischen Wissenschaften. Springer-Verlag, 1985.

[7] Andrew Butterfield. Reasoning about I/O in functional programs. In *Proceedings of the 4th Central European Functional Programming School*, CEFP'11, pages 93–141, Berlin, Heidelberg, 2012. Springer-Verlag.

[8] Venanzio Capretta. General recursion via coinductive types. *Logical Methods in Computer Science*, 1(2), 2005.

[9] Pietro Cenciarelli and Eugenio Moggi. A syntactic approach to modularity in denotational semantics. In *Proceedings of the 5th Biennial Meeting on Category Theory and Computer Science, CTCS 93, CWI Technical Report*, Amsterdam, The Netherlands, 1993.

[10] Calvin C. Elgot. Monadic computation and iterative algebraic theories. In *Logic Colloquium '73, Proc., Bristol 1973, 175-230*, 1975.

[11] Andrzej Filinski and Kristian Støvring. Inductive reasoning about effectful data types. In *Proceedings of the 12th ACM SIGPLAN International Conference on Functional Programming*, ICFP '07, pages 97–110, New York, NY, USA, 2007. ACM.

[12] Neil Ghani and Tarmo Uustalu. Coproducts of ideal monads. *Theoretical Informatics and Applications*, 38(4):321–342, 2004.

[13] Peter Hancock and Anton Setzer. Guarded induction and weakly final coalgebras in dependent type theory. In L. Crosilla and P. Schuster, editors, *From Sets and Types to Topology and Analysis. Towards Practicable Foundations for Constructive Mathematics*, pages 115 – 134, Oxford, 2005. Clarendon Press.

[14] Ichiro Hasuo, Bart Jacobs, and Ana Sokolova. Generic trace semantics via coinduction. *Logical Methods in Computer Science*, 3(4), 2007.

[15] Graham Hutton and Diana Fulger. Reasoning about effects: Seeing the wood through the trees. In *Proceedings of the Symposium on Trends in Functional Programming*, Nijmegen, The Netherlands, May 2008.

[16] Martin Hyland, Gordon D. Plotkin, and John Power. Combining effects: Sum and tensor. *Theoretical Computer Science*, 357(1-3):70–99, 2006.

[17] Simon L. Peyton Jones and Philip Wadler. Imperative functional programming. In Mary S. Van Deusen and Bernard Lang, editors, *Symposium on Principles of Programming Languages, Charleston, South Carolina, USA*, pages 71–84. ACM Press, 1993.

[18] Stefan Milius. Completely iterative algebras and completely iterative monads. *Information and Computation*, 196:1–41, 2005.

[19] Stefan Milius and Lawrence S. Moss. The category-theoretic solution of recursive program schemes. *Theoretical Computer Science*, 366(1-2):3–59, 2006.

[20] Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.

[21] Philip S. Mulry. Lifting theorems for Kleisli categories. In Stephen D. Brookes, Michael G. Main, Austin Melton, Michael W. Mislove, and David A. Schmidt, editors, *Mathematical Foundations of Programming Semantics, 9th International Conference, New Orleans, LA, USA*, volume 802 of *Lecture Notes in Computer Science*, pages 304–319. Springer, 1993.

[22] Keiko Nakata and Tarmo Uustalu. Trace-based coinductive operational semantics for While. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *TPHOLs*, volume 5674 of *Lecture Notes in Computer Science*, pages 375–390. Springer, 2009.

[23] Hanne Riis Nielson and Flemming Nielson. *Semantics with applications: a formal introduction.* John Wiley & Sons, Inc., New York, NY, USA, 1992.

[24] Maciej Piróg and Jeremy Gibbons. Tracing monadic computations and representing effects. In James Chapman and Paul Blain Levy, editors, *Proceedings Fourth Workshop on Mathematically Structured Functional Programming, Tallinn, Estonia, 25 March 2012*, volume 76 of *Electronic Proceedings in Theoretical Computer Science*, pages 90–111. Open Publishing Association, 2012.

[25] Gordon D. Plotkin. Adequacy for infinitary algebraic effects (abstract). In *3rd Conference on Algebra and Coalgebra in Computer Science, CALCO 2009, Udine, Italy*, pages 1–2, 2009.

[26] Jan J. M. M. Rutten. A note on coinduction and weak bisimilarity for While programs. *Theoretical Informatics and Applications*, 33(4/5):393–400, 1999.

[27] Michael B. Smyth and Gordon D. Plotkin. The category-theoretic solution of recursive domain equations. *SIAM Journal on Computing*, 11(4):761–783, 1982.

[28] Wouter Swierstra and Thorsten Altenkirch. Beauty in the beast: A functional semantics of the awkward squad. In *Haskell '07: Proceedings of the ACM SIGPLAN Workshop on Haskell*, pages 25–36, 2007.

Automata-Theoretic Semantics of Idealized Algol with Passive Expressions

Uday S. Reddy[1]

School of Computer Science
University of Birmingham
Birmingham, U.K

Abstract

Passive expressions in Algol-like languages represent computations that read the state but do not modify it. The need for such read-only computations arises in programming logics as well as in concurrent programming. It is also a central facet in Reynolds's Syntactic Control of Interference. Despite its importance and essentially basic character, capturing the notion of passivity in semantic models has proved to be difficult. In this paper, we provide a new model of passive expressions using an automata-theoretic framework recently proposed by the author. The central idea is that the store of a program is viewed as an abstract form of an automaton, with a representation of its states as well as state transitions. The framework allows us to combine the strengths of conventional state-based models and the more recent event-based models to synthesize new "automata-based" models. Once this basic framework is set up, relational parametricity does the job of identifying passive computations.

Keywords: Idealized Algol, Relational parametricity, Functor categories, Reflexive graphs, Algebraic automata theory.

1 Introduction

We expect that denotational semantic models of programming languages provide a rigorous *conceptual* foundation for reasoning about programs. In devising such models, one is faced with the challenge of how best to capture the intuitions the programmers possess in understanding computations and incorporate them in a rigorous theoretical framework.

The traditional models for imperative programming languages, dating back to those of Scott and Strachey, are *state-based*. These models envisage that programs operate on a store which goes through states. Commands are interpreted as *functions* from states to states, factoring out all the internal state manipulation details carried out by them. Thus, these models may be regarded as being *extensional* in their treatment of the store. Examples of such models include the original

[1] Email: u.s.reddy@bham.ac.uk

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.com/locate/entcs

models due to Scott and Strachey [40], the functor category models initiated by Reynolds [29, 36, 41] and their refinements using relational parametricity [25, 26].

In more recent developments, an alternative *event-based* approach for modeling computations has come to the fore. These models eschew any notion of store or state. They view commands as *processes* that interact with the individual storage variables via interaction events. The process-based view of commands exposes all their internal state manipulation details and makes the models *intensional*. On balance, however, the data abstraction and information hiding aspects of storage variables are captured more directly in these models. They are also able to model the intensional aspects of the computations such as the idea of "irreversible state change," leading to strong full abstraction results. Examples of such event-based models include the process calculus models due to Milner and Hoare [16, 21], Brookes's trace models [8], the author's object-based models [20, 24, 31, 32] and the games models [2].

The difference between extensional and intensional models becomes manifest in reasoning about program equivalences such as:

$$\mathbf{gv}(x) \implies (x := x + 1;\ x := x + 1) \equiv (x := x + 2) \tag{1}$$

where $\mathbf{gv}(x)$ represents the condition that x is a "good variable" obtained by variable allocation. Extensional models satisfy such equivalences because they capture the net effect of commands on the state, whereas intensional models do not. [2] However, the treatment of data abstraction (local variables) and irreversibility of state change is problematic in extensional models.

In an effort to combine the advantages of state-based and event-based models, we recently initiated a new approach using an automata-theoretic view of the store [33, 34]. The store is viewed as an automaton with an explicit representation of the states as well as the state transitions. The use of states allows an extensional treatment of commands and the use of state transitions captures some aspects of the modelling available in event-based models. We showed that several program equivalences of third-order types that could not be validated in the pure state-based models are valid in this setting.

In this paper, we take a further step in the development of the automata-theoretic model by modelling *passive expressions*, as per Reynolds's original Idealized Algol [36]. Passive expressions read the storage variables to compute values, but they do not alter the store. Typical programming languages allow side-effects in expressions for practical reasons, leaving it to the programmer to use them judiciously. [3] However, passive expressions form an integral part of program reasoning. For instance, in Hoare Logic [15], expressions can be embedded in logical assertions, where any side effects can lead to an entirely incoherent formalism. In concurrent programming, passive expressions form an important tool for sharing resources across processes. Various program reasoning systems, ownership type

[2] One might find it surprising that the intensional models, e.g., games models, fail to be "extensional" despite being fully abstract. The explanation is that full abstraction only guarantees the satisfaction of *unconditional* equivalences which seem inadequate to capture extensionality of state-manipulation.

[3] The evaluation order of expressions is often left unspecified or under-specified, so that an uncontrolled use of expression side effects is not a practical proposition in any case.

systems etc. incorporate explicit annotation for "read-only" or "immutable" variables, which depend on notions of passive usage [19, 22]. In particular, the use of "fractional permissions" is an advanced mechanism to capture the passive use of storage, currently an active area of research [6, 7, 35].

Modeling passivity in extensional models is a significant challenge because passivity appears to be an *intensional phenomenon*: what a computation does internally in order to produce its results. If we think of modelling expressions as extensional functions of type $\mathit{State} \to \mathit{Value}$, we have no handle on what such a computation might do. It might internally calculate a new state (which means a *state change* in computational terms), and do further computations within the new state to deliver the result. The new state is eventually discarded, and the expression would have had a "temporary side effect." This kind of a phenomenon can be captured syntactically by a "snap back" combinator of the form:

$$\mathbf{do}\ C\ \mathbf{result}\ E$$

which means "execute the command C and return the value of expression E, discarding the effects of C." The presence of such a snap back combinator in the semantic models breaks intuitive program equivalences. For instance, consider the equivalence: [4]

$$\begin{array}{l} \mathbf{if}\ (\mathrm{deref}\ x = 0)\ \mathbf{then}\ f(\mathrm{deref}\ x)\ \mathbf{else}\ 2\ \equiv \\ \qquad\qquad \mathbf{if}\ (\mathrm{deref}\ x = 0)\ \mathbf{then}\ f(0)\ \mathbf{else}\ 2 \end{array} \tag{2}$$

where f is a function procedure taking an expression argument. Since f is called only in the case where x is 0, giving it 0 as the argument instead of (deref x) should give equivalent results. However, in a semantic model that contains the snap back operator, there are functions f that break this reasoning, for example:

$$f \;=\; \lambda e.\, \mathbf{do}\ x := x + 1\ \mathbf{result}\ e$$

With this function f, the left hand side evaluates to 1 whereas the right hand side evaluates to 0. Virtually all extensional models in the literature, with the exception of the Tennent's model [41], have such snap back combinators.

We get around the difficulty by viewing the store as an automaton, which has an explicit representation of its states $\mathcal{Q}_X$ as well as its allowed state transformations $\mathcal{T}_X$. The expression type may then be thought of as a type constructor parameterized by both the components of the automaton:

$$\mathrm{Exp}(\mathcal{Q}_X, \mathcal{T}_X) = [\mathcal{Q}_X \to \mathit{Value}]$$

All computations are expected to be *parametric* [11, 26, 30, 38], i.e., they are interpreted by parametrically polymorphic families of the form:

$$\forall \mathcal{Q}_X, \mathcal{T}_X.\ F(\mathcal{Q}_X, \mathcal{T}_X) \to \mathrm{Exp}(\mathcal{Q}_X, \mathcal{T}_X)$$

[4] Imperative programming languages usually involve an implicit coercion that allows a storage variable to be treated as an expression that reads its contents. We represent this coercion as "deref" for clarity of exposition. Recall also that Idealized Algol is a call-by-name typed lambda calculus. So, the argument is passed by name in $f(\mathrm{deref}\ x)$.

where $F(\mathcal{Q}_X, \mathcal{T}_X)$ represents the semantic type of the free identifiers. Since the result type $\textsc{Exp}(\mathcal{Q}_X, \mathcal{T}_X)$ is independent of the $\mathcal{T}_X$ components, parametricity says that the family should behave the "same way," no matter what type $\mathcal{T}_X$ is employed (subject to some constraints). In particular, it should produce the same results if $\mathcal{T}_X$ is replaced by a trivial collection of state transformations, such as the one with just the identity transformation and its possibly diverging approximations. It then follows that the expression computation cannot cause any state changes, not even temporary ones. Thus passivity is captured in an intuitively satisfactory form.

The definition of this model builds on two technical innovations from our past work (joint with B. P. Dunphy). The first is the categorical axiomatization of relational parametricity presented in [11]. Since the overall structure is that of a category-theoretic possible world model, as pioneered by Reynolds [36], a categorical treatment of parametricity is needed to build the model we seek. O'Hearn and Tennent [26] initiated the building-in of relational parametricity into categories. However, their model does not have the requisite axioms, and snap back operators are present in their model. Our axiomatization is based on the notion of *fibrations*, well-studied in category theory [14, 18], using which strong representation results were obtained in [11]. Its employment here gives further evidence of its power. The second innovation is the automata-theoretic modeling of the store presented in [33, 34]. In retrospect, this view of the store was already implicit in Reynolds's first functor category model [36]. However, the automata-theoretic intuitions behind his model were not recognized and subsequently ignored in all further work on functor category models. Our model seems to have been the first work that builds on Reynolds's ideas. In the present work, we generalize the automata-theoretic model in a significant way, which parallels Tennent's generalization of the Oles model [41], in order to capture the seemingly intensional phenomenon of passivity. In doing such a generalization, it is easy to go too far to the other way, i.e., to make the model so intensional that the equivalence (1) fails. Tennent's model, in fact, breaks this equivalence. (Contrary to expectation, the equivalence cannot be derived in Specification Logic.) We aim to achieve a delicate balance of intensional effects and extensionality in the present paper.

Results

The main contribution of this paper is to provide a denotational model of Idealized Algol that satisfactorily models passivity while being extensional. In particular, this means that passive expressions do not have side effects, not even temporary ones. In the main body of the paper, we do this for a language without divergence, but treat it in such a way that it generalizes to divergence. The issues of divergence are then briefly mentioned in Sec. 5. The treatment without divergence is also novel in that it is the first model of passivity that is able to deal with a language without divergence. All the previous models [2, 32, 41] depend on the presence of divergence for modeling passivity. However, intuitively, passivity is independent of the issues of divergence. Our treatment is able to decouple the two issues.

We can explain the contribution in terms of the accuracy gained at first-order types [24, 32]. In the absence of divergence:

- Morphisms of type **com** → **com** should be isomorphic to natural numbers. They are all expressible by closed terms of the form $\lambda c.\, c^n$ where c^n means an n-fold sequential composition $c; \ldots ; c$. The model of [34] has this property.

- Morphisms of type **com** → $\mathbf{exp}[\delta]$ should be constant functions. They are expressible by closed terms of the form $\lambda c.\, E$ for closed expression terms E. The present model has this property.

2 Semantic Framework

The semantic framework used in this paper is that of a category-theoretic possible worlds model, as advocated by Reynolds [36]. That means that the *types* of the programming language are interpreted as *type constructors* parameterized by "store shapes" (formally *functors*). For example, $\textsc{Exp}(X)$ represents the collection of expression meanings appropriate for stores of shape X, $\textsc{Com}(X)$ represents the collection of command meanings appropriate for stores of shape X *etc*. The store shapes must form a category where morphisms $f : X \to Y$ represent ways in which a store Y may be regarded as a "future world" of X (typically by allocating additional storage locations). It might in fact be helpful to think of such a morphism as a "function" going in the reverse direction, $f^\sharp : Y \to X$, capturing a way of "extracting" an X-typed store from a Y-typed one. The type functors, naturally, must map such morphisms to functions. For example, $\textsc{Exp}(f \colon X \to Y)$ denotes a function that allows us to convert an expression on X-typed stores to one on Y-typed stores, which is possible because X-typed stores can be extracted from Y-typed stores.

In addition to morphisms, we consider abstract "logical relations" between stores, used for formulating the uniformity conditions of relational parametricity. For every pair of stores X and X', we have a notion of logical relations $R : X \leftrightarrow X'$ and a notion of morphisms preserving such relations, which is written diagrammatically as a "square":

$$\begin{tikzcd}
X \arrow[r, "f"] \arrow[d, leftrightarrow, "R"'] & Y \arrow[d, leftrightarrow, "S"] \\
X' \arrow[r, "f'"] & Y'
\end{tikzcd} \tag{3}$$

and textually as $f\ \big[R \to S\big]\ f'$. (The textual notation depends on the fact that all the structures we consider in this paper are *relational*, i.e., given f, f', R and S, there is at most one square of the above shape. Therefore $R \to S$ may be regarded as a normal set-theoretic relation between hom-sets $X \to Y$ and $X' \to Y'$.) The type functors also map such logical relations between stores to relations between values, e.g., $\textsc{Exp}(R) : \textsc{Exp}(X) \leftrightarrow \textsc{Exp}(X')$, and "squares" of the form (3) to relation-preservation squares between functions.

$$\begin{tikzcd}
\textsc{Exp}(X) \arrow[r, "\textsc{Exp}(f)"] \arrow[d, leftrightarrow, "\textsc{Exp}(R)"'] & \textsc{Exp}(Y) \arrow[d, leftrightarrow, "\textsc{Exp}(S)"] \\
\textsc{Exp}(X') \arrow[r, "\textsc{Exp}(f')"] & \textsc{Exp}(Y')
\end{tikzcd}$$

Formally, the four components: store shapes, morphisms between store shapes, logical relations between store shapes and squares between them, form a *reflexive graph* of categories. Further, they satisfy additional axioms laid out in [11] to form a *parametricity graph.* Formal definitions describing the structure may be found in the Appendix.

In addition to the reflexive graph of store shapes, which will described in the remainder of this section, we also make use of the reflexive graph **Set**, whose objects and morphisms are sets and functions, "logical relations" are set-theoretic relations $R \subseteq A \times A'$ and "squares" $f \; \big[R \to S\big] \; f'$ represent the facts $\forall a, a'.\, a \; \big[R\big] \; a' \implies f(a) \; \big[S\big] \; f'(a')$. This reflexive graph also satisfies the additional requirements of parametricity graphs.

Reynolds monoids

We choose to model stores as an abstract form of automata similar to those studied in algebraic automata theory [12, 17]. Each such automaton has: [5]

- a set of states $\mathcal{Q}_X$,
- a monoid of allowed state transformations $\mathcal{T}_X \subseteq [\mathcal{Q}_X \to \mathcal{Q}_X]$ (containing the identity transformation, written as 1_X, and closed under sequential composition $a \cdot b$), and
- an operation $\mathrm{read}_X : (\mathcal{Q}_X \to \mathcal{T}_X) \to \mathcal{T}_X$ defined by $\mathrm{read}_X\, p = \lambda x.\, p\, x\, x$.

A structure of this form is called a *Reynolds monoid.*

The read_X operation was proposed by Reynolds [36], who called it "diagonalization." To see the motivation for it, consider interpreting a command of the form **if** p **then** c_1 **else** c_2. This command reads the state to compute the boolean expression p and, depending on the result, executes either c_1 or c_2, which are both expected to denote allowed transformations. The overall transformation must in turn be an allowed transformation. Since the command chooses one among several allowed transformations based on the current state, we expect that all such choices should be allowed transformations. The existence of the read_X operation ensures this property. If a given automaton $(\mathcal{Q}_X, \mathcal{T}_X)$ does not have a read_X operation, additional transformations can be added to $\mathcal{T}_X$ to obtain a Reynolds monoid. We call it the "read-closure" of the original automaton.

As examples of Reynolds monoids, consider a store Z with

$$\mathcal{Q}_Z = \mathit{Int} \qquad\qquad \mathcal{T}_Z = \{\, a \colon \mathit{Int} \to \mathit{Int} \mid a(z) \geq z \,\}$$

This store contains a single integer variable and allows it to be increased during computations (but not decreased). A "passive" store W has some state set, but only the do-nothing transformation $\mathcal{T}_W = \{1_W\}$. For every store X, there is a corresponding *passive store of* X, denoted X_0, which has the same state set as that of X and the trivial state transformations $\mathcal{T}_X = \{1_X\}$.

The automata used in [33, 34, 36], called Reynolds *transformation monoids*, have an additional element of structure:

- a monoid action of type $\alpha_X : \mathcal{T}_X \to (\mathcal{Q}_X \to \mathcal{Q}_X)$ which represents a way of

[5] For reasons of exposition, we will ignore the issues of divergence in the main body of the paper. However, see Sec. 5 for the extensions needed for divergence.

"running" a transformation on the states.

Here, we drop this operation, obtaining generality in the structures as well as the corresponding morphisms and logical relations. The justification for the generalization is that states in imperative programs are "abstract," available for inspection only by other commands but not by external interfaces. By requiring that logical relations only preserve the read operation, and not the monoid action, we obtain more relations, which gives a stronger parametricity criterion. This generalization is used in our intuitive argument in the Introduction. In order to replace a state transformation component $\mathcal{T}_X$ by a trivial one, we need to allow for the possibility that the new transformations have a *different effect* on the state than the ones we are replacing. This generalization is crucial for modelling passivity.

A *logical relation* of Reynolds monoids $R : X \leftrightarrow X'$ is a pair (R_q, R_t) where

- $R_q : \mathcal{Q}_X \leftrightarrow \mathcal{Q}_{X'}$ is a normal relation of sets, and
- $R_t : \mathcal{T}_X \leftrightarrow \mathcal{T}_{X'}$ is a monoid relation (compatible with identity transformation and composition),

such that $\mathrm{read}_X \; \left[(R_q \to R_t) \to R_t\right] \; \mathrm{read}_{X'}$. The identity logical relation of a Reynolds monoid X is $I_X = (\Delta_{\mathcal{Q}_X}, \Delta_{\mathcal{T}_X})$ consisting of the diagonal relations on both the state sets and the transformations.

A *morphism* of Reynolds monoids $f : X \to Y$, representing a way of expanding a "current world" X to a "future world" Y, is a pair (f_q, f_t):

$$f_q : \mathcal{Q}_X \leftarrow \mathcal{Q}_Y \qquad\qquad f_t : \mathcal{T}_X \to \mathcal{T}_Y$$

where f_q is a set-theoretic function and f_t is a monoid morphism satisfying $f_t(\mathrm{read}_X(p)) = \mathrm{read}_Y(f_t \circ p \circ f_q)$. The condition on read can also be written using relational notation as $\mathrm{read}_X \; \left[\langle f_q \to f_t\rangle \to \langle f_t\rangle\right] \; \mathrm{read}_Y$. The mutually opposite direction of the two functions f_q and f_t may be understood by thinking of morphisms $f : X \to Y$ as ways of extracting X-typed stores from Y-typed stores (i.e., the "current world" from the "future world"). To do such extraction, it should be possible to interpret all Y-typed states as X-typed states, which is done by the function f_q. The transformations of the stores, on the other hand, are invoked by the computational environment in which the store is embedded. If the environment requests a transformation a on the X-typed store, the simulation must interpret it as a transformation $f_t(a)$ of the Y-typed store. This kind of bidirectional information flow is now well familiar from games semantics.

A *square* of Reynolds monoids is defined as a pair of relation-preservation squares (for sets and monoids, respectively):

$$\begin{tikzcd} X \arrow[r, "f"] \arrow[d, leftrightarrow, "R"] & Y \arrow[d, leftrightarrow, "S"'] \\ X' \arrow[r, "f'"] & Y' \end{tikzcd} \iff \begin{tikzcd} \mathcal{Q}_Y \arrow[r, "f_q"] \arrow[d, leftrightarrow, "S_q"] & \mathcal{Q}_X \arrow[d, leftrightarrow, "R_q"'] \\ \mathcal{Q}_{Y'} \arrow[r, "f'_q"] & \mathcal{Q}_{X'} \end{tikzcd} \wedge \begin{tikzcd} \mathcal{T}_X \arrow[r, "f_t"] \arrow[d, leftrightarrow, "R_t"] & \mathcal{T}_Y \arrow[d, leftrightarrow, "S_t"'] \\ \mathcal{T}_{X'} \arrow[r, "f'_t"] & \mathcal{T}_{Y'} \end{tikzcd}$$

Note that the squares on the right (in **Set** and **Mon**) have their standard meaning:

$$\begin{gathered}\forall y \in \mathcal{Q}_Y,\, y' \in \mathcal{Q}_{Y'}.\; y \; \big[S_q\big] \; y' \implies f_q(y) \; \big[R_q\big] \; f'_q(y') \\ \forall a \in \mathcal{T}_Y,\, a' \in \mathcal{T}_{Y'}.\; a \; \big[R_t\big] \; a' \implies f_t(a) \; \big[S_t\big] \; f'_t(a')\end{gathered}$$

This data constitutes a *reflexive graph category* **RM** of Reynolds monoids.

Parametricity graphs

The so-called "parametricity graphs" are reflexive graphs of categories satisfying certain axioms, proposed in [11] for modelling relational parametricity. A parametricity graph is a reflexive graph that is:

- *relational*, i.e., there is at most one square of a given shape,
- *fibred* with chosen cleavage, and
- satisfies the *identity condition*, i.e., whenever $f \; \big[I_A \to I_B\big] \; g$, we have $f = g$.

The "relational" condition essentially simplifies the theory. The "identity condition" gives semantics to the identity logical relations. The "fibred" condition is a categorical treatment of inverse images of relations. (See Appendix for full definitions.) Given f, f' and S as in the square on the left below, there must be a pre-image $\langle f, f'\rangle^* \, S$ that can fill the dotted line in a universal way:

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[d, leftrightarrow, dotted, "\langle f, f'\rangle^* \, S"'] & B \arrow[d, leftrightarrow, "S"] \\ A' \arrow[r, "f'"] & B' \end{tikzcd} \qquad \begin{tikzcd} A \arrow[r, "f"] \arrow[d, leftrightarrow, "R"'] & B \arrow[d, leftrightarrow, dotted, "\langle f, f'\rangle_! \, R"] \\ A' \arrow[r, "f'"] & B' \end{tikzcd}$$

Squares of this form are called *cartesian squares*. The dual form of squares, *co-cartesian squares*, give "direct images" $\langle f, f'\rangle_! \, R$. The reflexive graph **Set** has both pre-images and direct images, given by:

$$\begin{aligned}\langle f, f'\rangle^* \, S &= \{\, (x, x') \mid f(x) \; \big[S\big] \; f'(x') \,\} \\ \langle f, f'\rangle_! \, R &= \{\, (f(x), f'(x')) \mid x \; \big[R\big] \; x' \,\}\end{aligned}$$

The reflexive graph **RM** is a parametricity graph. It satisfies the identity condition because it is obtained by putting together **Set** and **Mon**, both of which satisfy the identity condition. It is fibred with chosen cleavage:

$$\langle f, f'\rangle^* \, S = (\langle f_q, f'_q\rangle_! \, S_q,\; \langle f_t, f'_t\rangle^* \, S_t)$$

Diagrammatically:

$$\begin{tikzcd} \mathcal{Q}_X \arrow[d, leftrightarrow, dotted, "\langle f_q, f'_q\rangle_! \, S_q"'] & \mathcal{Q}_Y \arrow[l, "f_q"'] \arrow[d, leftrightarrow, "S_q"] \\ \mathcal{Q}_{X'} & \mathcal{Q}_{Y'} \arrow[l, "f'_q"] \end{tikzcd} \qquad \begin{tikzcd} \mathcal{T}_X \arrow[r, "f_t"] \arrow[d, leftrightarrow, dotted, "\langle f_t, f'_t\rangle^* \, S_t"'] & \mathcal{T}_Y \arrow[d, leftrightarrow, "S_t"] \\ \mathcal{T}_{X'} \arrow[r, "f'_t"'] & \mathcal{T}_{Y'} \end{tikzcd}$$

RM is not cofibred in general, but it does have some useful co-cartesian squares which will be put to use in the next section.

$$\begin{array}{ll}
\textsc{Com}(X) = \mathcal{T}_X & \textsc{Com}(R) = R_t \\
\textsc{Exp}_\delta(X) = [\mathcal{Q}_X \to [\![\delta]\!]] & \textsc{Exp}_\delta(R) = [R_q \to \Delta_{[\![\delta]\!]}] \\
\textsc{Var}_\delta(X) = \textsc{Exp}_\delta(X) \times [[\![\delta]\!] \to \textsc{Com}(X)] & \textsc{Var}_\delta(R) = \textsc{Exp}_\delta(R) \times [\Delta_{[\![\delta]\!]} \to \textsc{Com}(R)] \\
(F \times G)(X) = F(X) \times G(X) & (F \times G)(R) = F(R) \times G(R) \\
(F \Rightarrow G)(X) = \forall_{h:Z \leftarrow X}\,[F(Z) \to G(Z)] & (F \Rightarrow G)(R) = \forall_{S \leftarrow R}\,[F(S) \to G(S)]
\end{array}$$

Fig. 1. Interpretation of Idealized Algol types

Type functors

To interpret the types of Idealized Algol we use functors of appropriate kind from **RM** to **Set**, as shown in Fig. 1. This formalizes the intuition mentioned in Introduction that types are interpreted as "type constructors" parameterized by the store automaton.

A reflexive graph-functor (RG-functor) $F : \mathbf{G} \to \mathbf{H}$ between reflexive graphs maps all four components of the reflexive graph (objects, morphisms, logical relations and squares) preserving their structure. A *PG-functor* is a reflexive graph-functor that also preserves the *cartesian squares* and, in particular, the chosen cleavage:

$$F(\langle f, f' \rangle^* S) = \langle Ff,\, Ff' \rangle^*(FS)$$

We also insist that the functors used for interpreting Idealized Algol preserve all the *co-cartesian squares* that exist in **RM**. The category of PG-functors of this kind is denoted $\mathcal{P}(\mathbf{RM})$.

The morphisms in $\mathcal{P}(\mathbf{RM})$ are transformations that preserve all morphisms (naturality) as well as all relations (parametricity). However, under the conditions of parametricity graphs, parametricity implies naturality [11, 30]. So, we simply call them *parametric transformations.*

Theorem 2.1 *If* $\mathbf{C}$ *is a parametricity graph, the category* $\mathcal{P}(\mathbf{C})$ *of PG-functors* $\mathbf{C} \to \mathbf{Set}$ *preserving co-cartesian squares is cartesian closed.*

Products are given pointwise: $(F \times G)(X) = F(X) \times G(X)$ and $(F \times G)(R) = F(R) \times G(R)$. Exponents are given as in presheaf categories: $(F \Rightarrow G)(X) = \forall_{h:Z \leftarrow X}[F(Z) \to G(Z)]$, where $\forall$ denotes the "parametric limit" (in **Set**) indexed by morphisms h originating from X [11]. Explicitly, the parametric limit consists of families of the form

$$\langle t_h \in [F(Z) \to G(Z)] \rangle_{h:X \to Z}$$

that are *parametric* in the sense that

$$h\ \big[I_X \to S\big]\ h' \Rightarrow t_h\ \big[F(S) \to G(S)\big]\ t_{h'}$$

Since F and G are PG-functors, such families are automatically natural [11]. The relation $\forall_{S \leftarrow R}[F(S) \to G(S)]$ relates two families $\langle t_h \rangle_{h:X \to Z}$ and $\langle t'_{h'} \rangle_{h':X' \to Z'}$ iff, for all logical relations $S : Z \leftrightarrow Z'$ and all h, h' of appropriate types:

$$h \; \big[R \to S\big] \; h' \Longrightarrow t_h \; \big[F(S) \to G(S)\big] \; t'_{h'}$$

□

This result is a mild generalization of that in [11]. It establishes that the cartesian closed structure is present even for functors that preserve co-cartesian squares.

3 Modeling Passivity

Intuitively, a computation is passive if it *reads* the state but carries out no state changes. Since our stores $X = (\mathcal{Q}_X, \mathcal{T}_X)$ have a state set component and a state transformation component, this means that passive computations should only depend on the $\mathcal{Q}_X$ components and be independent of the $\mathcal{T}_X$ components.

We use relational parametricity to formalize these concepts. Call a logical relation $R : X \leftrightarrow X'$ a *transformer relation* if its state set component is the diagonal relation: $R_q = \Delta_{\mathcal{Q}_X}$. There are no constraints on the transformation component of the logical relation (except those imposed by Reynolds monoids).

Definition 3.1 *Given a PG-functor F in $\mathcal{P}(\mathbf{RM})$ and a store X, a value $d \in FX$ is said to be* passive *if, for all transformer relations $R : X \leftrightarrow X$, d is related to itself by FR, i.e., $d \; \big[FR\big] \; d$.*

This accords with our intuition. Since transformer relations keep the state set components of worlds fixed but allow the transformation components to vary, if a value is related to itself under all such variations, it must be independent of the transformation components. It is easy to see that all values $e \in \textsc{Exp}(X)$ are passive, as one would expect. On the other hand, in $\textsc{Com}(X)$, a value a is passive if and only if $a \; \big[R_t\big] \; a$ for all transformer relations R. This is only possible if $a = 1_X$, the do-nothing state transformation. (When we consider divergence, the passive command values include all approximations of 1_X.)

A PG-functor itself may be regarded as a passive functor if all its values are passive (for all stores X). We require this uniformly for all stores X.

Definition 3.2 *A PG-functor F is said to be* passive *if, for all transformer relations $R : X \leftrightarrow X$, $FR = \Delta_{FX}$.*

Note that $\textsc{Exp}$ is a passive functor, and $\textsc{Com}$ is not. However, $\textsc{Com}$ has a passive subfunctor, denoted $\wp\textsc{Com}$, which includes 1_X at every store shape X. We examine how to characterize the passive subfunctors.

Passivity monomorphism

Recall that, for every store X, there is a corresponding passive store X_0, which has the same state set as X but has only trivial state transformations $\mathcal{T}_{X_0} = \{1_X\}$. Since X_0 allows no state changes, we expect that all values $d \in FX_0$ are passive (for all PG-functors F).

There is a passivity monomorphism $p_X : X_0 \rightarrowtail X$ given by the identity on state sets and the injection $T_{X_0} \hookrightarrow \mathcal{T}_X$. We will argue that this monomorphism is preserved by all PG-functors.

Definition 3.3 *A morphism $m : A \to B$ in a parametricity graph is a* cartesian monomorphism *if the following square is cartesian:*

$$\begin{tikzcd}
A \arrow[r, "m"] \arrow[d, leftrightarrow, dotted, "I_A"'] & B \arrow[d, leftrightarrow, "I_B"] \\
A \arrow[r, "m"] & B
\end{tikzcd}$$

In other words, $I_A = \langle m, m \rangle^* \, I_B$.

Cartesian monomorphisms are monomorphisms. Moreover, since PG-functors preserve all cartesian squares, they preserve cartesian monomorphisms as well.

It is easy to see that the passivity monomorphisms $p_X : X_0 \rightarrowtail X$ are cartesian. Hence, for all PG-functors F, their images Fp_X are (cartesian) monomorphisms. This means that FX_0 is always a subobject of FX. Under the assumption that F preserves co-cartesian squares in addition to cartesian squares, we can show that all passive values of FX are contained within the image of FX_0 under Fp_X.

Lemma 3.4 *If F is a PG-functor that preserves co-cartesian squares, then a value $d \in FX$ is passive if and only if there exists $d_0 \in FX_0$ such that $Fp_X(d_0) = d$.*

The "only if" direction is based on the fact that every transformer relation R has the square shown on the left below:

$$\begin{tikzcd}
X_0 \arrow[r, "p_X"] \arrow[d, leftrightarrow, "I_{X_0}"'] & X \arrow[d, leftrightarrow, "R"] & & FX_0 \arrow[r, "Fp_X"] \arrow[d, leftrightarrow, "I_{FX_0}"'] & FX \arrow[d, leftrightarrow, "FR"] \\
X_0 \arrow[r, "p_X"] & X & & FX_0 \arrow[r, "Fp_X"] & FX
\end{tikzcd}$$

As the PG-functor F maps it to the square on the right, all the values in the image under $Fp_X : FX_0 \to FX$ are related to themselves by FR. Hence all such values are passive. For the "if" direction, we use the co-cartesian square shown on the left below:

$$\begin{tikzcd}
X_0 \arrow[r, "p_X"] \arrow[d, leftrightarrow, "I_{X_0}"'] & X \arrow[d, leftrightarrow, dotted, "\varrho_X"] & & FX_0 \arrow[r, "Fp_X"] \arrow[d, leftrightarrow, "I_{FX_0}"'] & FX \arrow[d, leftrightarrow, dotted, "F\varrho_X"] \\
X_0 \arrow[r, "p_X"] & X & & FX_0 \arrow[r, "Fp_X"] & FX
\end{tikzcd}$$

where $\varrho_X : X \leftrightarrow X$ is given by $(\varrho_X)_q = \Delta_{\mathcal{Q}_X}$ and $(\varrho_X)_t = \{(1_X, 1_X)\}$. Since F preserves co-cartesian squares, this implies that all passive values of FX are contained within the image of Fp_X.

Passivity retractions

The passivity monomorphisms have retractions, i.e., morphisms $r_X : X \to X_0$ such that $p_X; r_X = \mathrm{id}_X$. They are defined by $(r_X)_q = \mathrm{id}_{\mathcal{Q}_X}$ and $(r_X)_t = \lambda a.\, 1_X$. The reverse composite $\varpi_X = r_X; p_X : X \to X$ is then idempotent (a "split" idempotent). We can characterize passive values and passive functors in terms of the retractions as follows:

Lemma 3.5 *An element $d \in FX$ is passive if and only if $F\varpi_X(d) = d$.*

Lemma 3.6 *A PG-functor F in $\mathcal{P}(\mathbf{RM})$ is passive if and only if $F\varpi_X = \mathrm{id}_{FX}$.*

These properties were used to define passivity by O'Hearn et al. [23]. While the split idempotents ϖ_X lead to elegant theory of bireflective subcategories [13, 23], they do not generalize to divergence. So, we also consider a relational variant of ϖ_X and state our results in terms of it. The logical relation $\xi_X : X \leftrightarrow X$ is given by $(\xi_X)_q = \Delta_{\mathcal{Q}_X}$ and $(\xi_X)_t = \{\, (a, 1_X) \mid a \in \mathcal{T}_X \,\}$. This relation satisfies an important property:

Lemma 3.7 *For all Algol type functors F, the relation $F\xi_X : FX \leftrightarrow FX$ has, as its domain, the entire set FX, and, as its range, the passive subset of FX.*

Passive subfunctors

If F is a PG-functor in $\mathcal{P}(\mathbf{RM})$, there is a passive PG-functor $\wp F$ in $\mathcal{P}(\mathbf{RM})$ defined by

$$\begin{aligned} (\wp F)X &= \text{the range of } Fp_X \\ (\wp F)f &= \text{the restriction of } Ff \text{ to } (\wp F)X \end{aligned}$$

This definition is based on the following property.

Lemma 3.8 *If F is a PG-functor in $\mathcal{P}(\mathbf{RM})$ and $f : X \to Y$ a morphism in* **RM** *then $Ff : FX \to FY$ sends passive values in FX to passive values in FY.*

Using Lemma 3.7, we can show the following result, establishing that passive functors form a reflective subcategory.

Theorem 3.9 *If F and P are Algol functors in $\mathcal{P}(\mathbf{RM})$ where P is passive, there is a bijection between the parametric transformations $F \to P$ and parametric transformations $\wp F \to P$.*

$$\mathit{Par}(F,\, P) \cong \mathit{Par}(\wp F,\, P)$$

Proof. If $t : F \to P$ is a parametric transformation, the corresponding $t_0 : \wp F \to P$ has components $(t_0)_X$ that are just the restriction of components t_X to passive values. We show that t_0 uniquely determines t. Since t preserves all logical relations, in particular the transformer relation $\xi_X : X \leftrightarrow X$, we have a relation-preservation square (in **Set**):

$$\begin{tikzcd} FX \arrow[r, "t_X"] \arrow[d, leftrightarrow, "F\xi_X"'] & PX \arrow[d, leftrightarrow, "P\xi_X"] \\ FX \arrow[r, "t_X"] & PX \end{tikzcd}$$

Since ξ_X is a transformer relation, $P\xi_X = \Delta_{PX}$. So, the above square means:

$$\forall d, d_0 \in FX.\, d \left[F\xi_X\right] d_0 \implies t_X(d) = t_X(d_0)$$

Since the range of $F\xi_X$ consists of only passive values (by Lemma 3.7), this means that t_X is uniquely determined by its action on passive values.

Conversely, given a parametric transformation $t_0 : \wp F \to P$, we obtain a parametric transformation $t = t_0 \circ Fr : F \to P$. Applying the construction above and noting that $\xi_X = \langle \varpi_X \rangle = \langle p_X \circ r_X \rangle$, we conclude that t uniquely determines t_0.

$$\begin{tikzcd}
FX \arrow[d, "r_{FX}"'] \arrow[dr, "t_X"] & \\
(\wp F)X \arrow[r, dashed, "(t_0)_X"] & PX
\end{tikzcd}$$

□

Theorem 3.10 *The passive subfunctor operator $\wp$ is in turn a functor $\wp : \mathcal{P}(\mathbf{RM}) \to \mathcal{P}(\mathbf{RM})$. It enjoys the isomorphisms:*

$$\begin{aligned}
\wp P &\cong P && \textit{for passive functors } P \\
\wp\textsc{Com} &\cong \mathbf{1} \\
\wp(F \times G) &\cong \wp F \times \wp G \\
F \Rightarrow P &\cong \wp F \Rightarrow P && \textit{for passive functors } P
\end{aligned}$$

Proof. If $t : F \to G$ is a parametric transformation, $\wp t : \wp F \to \wp G$ is just the restriction t_0 of t that acts on passive values. The first isomorphism is, in fact, an equality $\wp P = P$, and follows from the fact that the passive subset of PX is the entire PX. For $\textsc{Com}$, 1_X is the only passive value in $\textsc{Com}(X)$. So, $\wp\textsc{Com}(X)$ is a singleton. For $F \times G$, note that $(F \times G)p_X = Fp_X \times Gp_X : FX_0 \times GX_0 \to FX \times GX$. So, (d, e) is in the range of $(F \times G)p_X$ iff d is in the range of Fp_X and e is in the range of Gp_X. The last isomorphism follows from the definition $(F \Rightarrow P)(X) = \forall_{h:Z \leftarrow X} [FZ \to PZ]$. Since $[FZ \to PZ]$ is isomorphic to $[(\wp F)Z \to PZ]$, we have the isomorphism. □

4 Applications

In this section, we examine the consequences of the theory developed in the previous sections.

Interpretation of Idealized Algol

Idealized Algol [36] is a simply typed lambda calculus (with call-by-name parameter passing) with basic types that support imperative computations.

The interpretation of types $\mathbf{exp}[\delta]$, $\mathbf{com}$, $\mathbf{var}[\delta]$, $\theta_1 \times \theta_2$ and $\theta_1 \to \theta_2$ is as PG-functors in $\mathcal{P}(\mathbf{RM})$, shown in Figure 1. For readability, we have used notation such as $\textsc{Exp}_\delta$ for $[\![\mathbf{exp}[\delta]]\!]$ etc.

The interpretation of a term M with typing:

$$x_1 : \theta_1, \ldots, x_n : \theta_n \vdash M : \theta$$

is a parametric transformation of type

$$[\![M]\!] : (\textstyle\prod_{x_i} [\![\theta_i]\!]) \to [\![\theta]\!]$$

This means that, for each store shape X, $[\![M]\!]_X$ is a function of type $(\prod_{x_i} [\![\theta_i]\!](X)) \to [\![\theta]\!](X)$ such that all relations are preserved, i.e., for any relation $R : X \leftrightarrow X'$, we

$\mathrm{equal} : \textsc{Exp}_\delta \times \textsc{Exp}_\delta \to \textsc{Exp}_{\mathbf{bool}}$	$\mathrm{equal}_X(e_1, e_2) = \lambda s.\, e_1(s) = e_2(s)$
$\mathrm{cond}^E : \textsc{Exp}_{\mathbf{bool}} \times \textsc{Exp}_\delta \times \textsc{Exp}_\delta \to \textsc{Exp}_\delta$	$\mathrm{cond}^E_X(e, e_1, e_2) = \lambda s.\, e(s) \to e_1(s);\ e_2(s)$
$\mathrm{skip} : \mathbf{1} \to \textsc{Com}$	$\mathrm{skip}_X(*) = 1_X$
$\mathrm{seq} : \textsc{Com} \times \textsc{Com} \to \textsc{Com}$	$\mathrm{seq}_X(a, b) = a \cdot b$
$\mathrm{cond}^C : \textsc{Exp}_{\mathbf{bool}} \times \textsc{Com} \times \textsc{Com} \to \textsc{Com}$	$\mathrm{cond}^C_X(e, a, b) = \mathrm{read}_X\ \lambda s.\, e(s) \to a;\ b$
$\mathrm{for} : \textsc{Exp}_{\mathbf{int}} \times \textsc{Com} \to \textsc{Com}$	$\mathrm{for}_X(e, a) = \mathrm{read}_X\ \lambda s.\, a^{e(s)}$
$\mathrm{deref} : \textsc{Var}_\delta \to \textsc{Exp}_\delta$	$\mathrm{deref}_X(d, a) = d$
$\mathrm{assign} : \textsc{Var}_\delta \times \textsc{Exp}_\delta \to \textsc{Com}$	$\mathrm{assign}_X((d, a), e) = \mathrm{read}_X\ \lambda s.\, a(e(s))$
$\mathrm{newvar} : (\textsc{Var}_\delta \Rightarrow \textsc{Com}) \to \textsc{Com}$	$\mathrm{newvar}_X(p) = (\lambda s.\, (s, \mathit{init}_\delta)) \cdot p[\iota_1](\mathrm{mkvar}{\uparrow}_V^{X \star V}) \cdot (\lambda(s, n).\, s)$ $\quad \text{where } V = (\llbracket\delta\rrbracket, T(\llbracket\delta\rrbracket)) \quad \mathrm{mkvar} = (\lambda n.\, n,\ \lambda k.\, \lambda n.\, k)$

Fig. 2. Primitive operators of Idealized Algol

have $\llbracket M \rrbracket_X \left[(\prod_{x_i} \llbracket \theta_i \rrbracket(R)) \to \llbracket \theta \rrbracket(R) \right] \llbracket M \rrbracket_{X'}$. To the extent that Idealized Algol is a simply typed lambda calculus, this interpretation is standard [11, 26].

$$\begin{aligned}
&\llbracket x \rrbracket_X(u) = u(x) \\
&\llbracket \lambda x \colon \theta.\, M \rrbracket_X(u) = \Lambda h \colon Z \leftarrow X.\, \lambda d \colon \llbracket \theta \rrbracket(Z).\, \llbracket M \rrbracket_Z(u{\uparrow}_X^Z[x \mapsto d]) \\
&\llbracket MN \rrbracket_X(u) = \llbracket M \rrbracket_X(u)[\mathrm{id}_X \colon X \to X](\llbracket N \rrbracket_X(u))
\end{aligned}$$

The parameter u may be thought of as an "environment" that provides values for the free identifiers, specifically in the given world X. The meaning of a lambda abstraction of type $\theta \to \theta'$ is in $(\llbracket \theta \rrbracket \Rightarrow \llbracket \theta' \rrbracket)(X)$, which consists of families of the form $\langle t_h \rangle_{h : Z \leftarrow X}$. Here, we are using notation "$\Lambda h : Z \leftarrow X$" borrowed from the polymorphic lambda calculus to express the parameterization by h. Note that the body of the abstraction $\lambda x \colon \theta.\, M$ is interpreted in the future world Z and the environment u is "upgraded" to this world. We use the mnemonic short-hand notation $a{\uparrow}_X^Z$ for the value $\llbracket \theta \rrbracket(f)(a)$ when $f : X \to Z$ is the morphism available in the context and θ is the type of a. Parametricity in Z is crucial for capturing the fact that $\llbracket M \rrbracket_Z$ does not directly access any information of the future world. In the interpretation of function application terms, we are again using the polymorphic lambda calculus notation to pass in the h argument, which is $\mathrm{id}_X : X \to X$.

The imperative operations can be defined as a set of primitive constants, a sample of which are shown in Fig. 2. Their interpretations should be mostly self-explanatory. We are using the notation $p \to v_1; v_2$ to denote conditional expressions in semantic meta-language. Note that Reynolds's read operation is used in interpreting conditional commands as well as assignment, both of which use the current state information to construct a state transformation. Variable are represented as pairs of operations: an expression-typed operation that dereferences the variable and an "acceptor" that, given a value, stores it in the variable. The "newvar" primitive allocates a new variable in the context of a store X. It defines a new piece of store V with the state set $\llbracket \delta \rrbracket$ and all state-transformations on it, denoted $T(\llbracket \delta \rrbracket)$. The "mkvar" construction provides the dereference-acceptor pair on this store. To add the store V to the existing store X, we use a tensor product on stores denoted

$\star$ [34]. The store $X \star Y$ is defined as the Reynolds monoid:

$$\begin{aligned} \mathcal{Q}_{X \star Y} &= \mathcal{Q}_X \times \mathcal{Q}_Y \\ \mathcal{T}_{X \star Y} &= \text{read-closure of } \{\, a \times b \mid a \in \mathcal{T}_X,\, b \in \mathcal{T}_Y \,\} \end{aligned}$$

This store has evident injections $\iota_1 : X \to X \star Y$ and $\iota_2 : Y \to X \star Y$.

Examples

In the first place, let us note that the snap back combinator (**do** C **result** E) is ruled out. To interpret it we would need a parametric transformation of the form:

$$\begin{aligned} &\mathrm{do} : \textsc{Com} \times \textsc{Exp} \to \textsc{Exp} \\ &\mathrm{do}_X(a, e) = \lambda s.\, e(a(s)) \end{aligned}$$

We can see that it is not parametric. For example, the preservation of the relation $\xi_X : X \leftrightarrow X$ requires

$$\begin{tikzcd} \textsc{Com}(X) \arrow[d, leftrightarrow, "\textsc{Com}(\xi_X)"'] & \times & \textsc{Exp}(X) \arrow[r, "\mathrm{do}_X"] \arrow[d, leftrightarrow, "\textsc{Exp}(\xi_X)"'] & \textsc{Exp}(X) \arrow[d, leftrightarrow, "\textsc{Exp}(\xi_X)"] \\ \textsc{Com}(X) & \times & \textsc{Exp}(X) \arrow[r, "\mathrm{do}_X"] & \textsc{Exp}(X) \end{tikzcd}$$

which says $e(a(s)) = e(1_X(s))$ for all $a \in \textsc{Com}(X)$, $e \in \textsc{Exp}(X)$ and states $s \in \mathcal{Q}_X$. (Note that $a \left[\textsc{Com}(\xi_X)\right] 1_X$ and $e \left[\textsc{Exp}(\xi_X)\right] e$.) Since $1_X(s) = s$, we are requiring $e(a(s)) = e(s)$. The condition would be violated, for example, if X has at least two states, say $\{0, 1\}$, and a causes a state change, perhaps by sending 0 to 1, and e returns the integer in the current state.

Consider the equivalence stated in the Introduction:

$$\textbf{if } (\mathrm{deref}\ x = 0) \textbf{ then } f(\mathrm{deref}\ x) \textbf{ else } 2 \;\equiv\; \textbf{if } (\mathrm{deref}\ x = 0) \textbf{ then } f(0) \textbf{ else } 2$$

This requires that, for all worlds X, values $(e, a) \in \textsc{Var}(X)$ and $f \in (\textsc{Exp} \Rightarrow \textsc{Exp})(X)$:

$$\big(\lambda s.\, (e\ s) = 0 \longrightarrow f[\mathrm{id}_X]\, e\, s;\ 2\big) \;=\; \big(\lambda s.\, (e\ s) = 0 \longrightarrow f[\mathrm{id}_X]\, \overline{0}\, s;\ 2\big)$$

Consider a relation given by

$$R_q = \{\, (s, s) \mid e\ s = 0 \,\} \qquad R_t = \{(1_X,\, 1_X)\}$$

Since $e \left[\textsc{Exp}(R)\right] \overline{0}$, we must have, for all states s such that $e\ s = 0$,

$$f[\mathrm{id}_X]\, e\, s \left[\Delta_{\mathit{Int}}\right] f[\mathrm{id}_X]\, \overline{0}\, s$$

Noting that Δ_{Int} is nothing but the equality relation, we have a proof of the equivalence.

A more interesting variant of the above equivalence is:

$$\begin{aligned} &\textbf{if } (\mathrm{deref}\ x = \mathrm{deref}\ y) \textbf{ then } f(x) \textbf{ else } 2 \;\equiv \\ &\qquad \textbf{if } (\mathrm{deref}\ x = \mathrm{deref}\ y) \textbf{ then } f(y) \textbf{ else } 2 \end{aligned}$$

where $f : \mathbf{var} \to \mathbf{exp}$. The difference from the previous example is that we are passing the function procedure f the entire variable (x or y) rather than just an expression dereferencing it. So, one might wonder if there is a possibility of f changing the given variable. We argue abstractly, using the results of Theorem 3.10.

$$\begin{aligned}
\textsc{Var} \Rightarrow \textsc{Exp} &\cong \wp\textsc{Var} \Rightarrow \textsc{Exp} \\
&= \wp(\textsc{Exp} \times (\mathit{Int} \to \textsc{Com})) \Rightarrow \textsc{Exp} \\
&\cong \wp\textsc{Exp} \times \wp(\mathit{Int} \to \textsc{Com})) \Rightarrow \textsc{Exp} \\
&\cong \textsc{Exp} \times (\mathit{Int} \to \wp\textsc{Com}) \Rightarrow \textsc{Exp} \\
&\cong \textsc{Exp} \times (\mathit{Int} \to \mathbf{1}) \Rightarrow \textsc{Exp} \\
&\cong \textsc{Exp} \times \mathbf{1} \Rightarrow \textsc{Exp} \\
&\cong \textsc{Exp} \Rightarrow \textsc{Exp}
\end{aligned}$$

For the third step, regard $\mathit{Int} \to \textsc{Com}$ as a product $\prod_{i \in \mathit{Int}} \textsc{Com}$ and use an infinitary version of the product isomorphism. The calculation shows that a function procedure that receives a variable argument only has the ability to use its deref operation.

5 Handling divergence

For modeling divergence, we use a strict function model similar to that described in [25, Sec. 6]. Define a parametricity graph of Reynolds monoids with divergence, denoted $\mathbf{RM}_\perp$, where "state sets" $\mathcal{Q}_X$ are flat cpo's and transformations are complete ordered submonoids of the strict function space $[\mathcal{Q}_X \multimap \mathcal{Q}_X]$, equipped with a read_X operation. The functions involved in morphisms are required to be strict and continuous, and the relations are required to be complete (pointed and directed-complete). These properties are inherited from $\mathbf{CPO}_\perp$ which is itself a parametricity graph [11]. The semantic category is that of functors $(\mathbf{RM}_\perp)^{\mathrm{op}} \to \mathbf{DCPO}$ that factor through $\mathbf{CPO}_\perp$. It is a result of Oles [28, 24] that such a category is cartesian closed.

The passive store X_0 of a store X has the same cpo of states as X but, as transformations, all approximations of the do-nothing transformation:

$$\mathcal{T}_{X_0} = \{\, a \mid a \sqsubseteq 1_X \,\}$$

(The intermediate approximations are included by read-closure.) The passivity monomorphism $p_X : X_0 \rightarrowtail X$ involves the obvious injection of the complete ordered monoid $\mathcal{T}_{X_0} \hookrightarrow \mathcal{T}_X$.

However, unlike in $\mathbf{RM}$, the passivity monomorphisms do *not* have retractions.

Lemma 5.1 *There is no morphism $r_X : X \to X_0$ such that $p_X ; r_X = \mathrm{id}_{X_0}$.*

The technical reason for the failure is that sequential composition of transformations is not preserved in going from X to X_0. It is possible for a composite transformation $a \cdot b$ to appear "passive" even if a and b are not passive. The solution adopted by Tennent [41] for this situation is that such a transformation $a \cdot b$

should diverge in X_0. However, this solution loses extensionality. The *way* in which the passive computation is expressed becomes significant.

Since we employ relational parametricity rather than naturality, we offer another solution. The relation $\xi_X : X \leftrightarrow X$ mentioned under *Passivity retractions* can be adapted to deal with divergence as follows:

$$\begin{aligned}(\xi_X)_q &= \Delta_{\mathcal{Q}_X} \\ (\xi_X)_t &= \{\, (a, a') \mid a \sqcap 1_X \sqsupseteq a' \,\}\end{aligned}$$

It may be verified that $(\xi_X)_t$ is a monoid relation and ξ_X itself is a Reynolds monoid relation. Lemma 3.7 continues to hold for this relation ξ_X. Consequently, we can obtain a weaker version of Theorem 3.9, which is adequate for semantic reasoning.

Theorem 5.2 *If F and P are Algol functors in $\mathcal{P}(\mathbf{RM}_\perp)$ where P is passive, every parametric transformation $t : F \to P$ is uniquely determined by its restriction $t_0 : \wp F \to P$, giving a natural monomorphism:*

$$Par(F,\, P) \rightarrowtail Par(\wp F,\, P)$$

The proof is the same as the first half of Theorem 3.9 because it is proved using the relation ξ. This means that computations of type $F \to P$ are uniquely determined by their restrictions to $\wp F \to P$. Hence, they cannot have side effects, not even temporary ones.

6 Related work

The model of Specification Logic, due to Tennent [41], was the first one to model passivity. The passivity aspects were further studied in [13, 23]. Tennent's model does not employ relational parametricity, relying on morphisms instead of relations to capture the uniformity conditions. Passivity and other intensional properties are modelled through a form of "what if" modeling. Morphisms in the category of stores include, not only those needed for interpreting the programming language, but additional ones that are used in the logical analysis (including the left inverses of passivity monomorphisms). It is decided whether a computation is passive by asking the question what would happen to it under a morphism that prohibits all state changes. If it remains the same, then it is regarded as passive; otherwise not. While intuitively appealing, this model has the unfortunate effect of becoming intensional (despite working in an extensional framework). Two program terms are equivalent only if they behave identically under all possible state change constraints. For example, the equivalence (1) is not valid in the model, for the reason that the left hand side of the equivalence would be undefined if state changes were constrained to those that preserve the even-ness of x, whereas the right hand side would continue to be defined.

O'Hearn and Reynolds [25] provide an account of irreversible state change for the command type and active expressions via a syntactic translation to the polymorphic linear lambda calculus. While the explanation of state change via a linearly used state object is intuitively appealing, it has no way to allow for passive expressions. O'Hearn and Reynolds do not provide any treatment of passive expressions in their paper, and it is generally believed that it is not possible to do so in a purely linear setting. See, for example, Wadler [43] where an extension of linear lambda calculus is proposed for modeling "read-only" uses. At a semantic level, O'Hearn and Reynolds use strict functions on pointed cpo's to model state change, as previously recommended by the present author. This modeling eliminates snap back effects at the command type in the presence of divergence. It is adopted here in the same manner. However, our modeling of irreversible state change works even in the absence of divergence and, so, linearity and strictness are not central to it.

As remarked in Introduction, event-based models are able to model passivity with relative ease. However, all such models are intensional and do not satisfy extensional equivalences like (1). The "Passivity and Independence" model of the author [31] was historically the first one where the reflective subcategory structure of passive types was discovered. These ideas were later incorporated in the coherent space model [32] and the games model [2]. These models represent passivity by "fiat." Out of all possible events, certain event are designated as "passive," and the reflective subcategory structure is imposed via an axiom. In other words, these models state what is passive (rather correctly, it turns out), but do not explain what it *means* for a computation to be passive. The criticism that such a treatment lacks *explanatory force*, offered to the author by P. W. O'Hearn, P. Panangaden and others, formed the main driver for further investigation, culminating in the present results.

The Yoneda embedding of the coherent space model in a functor category shown in [24] bears a close intuitive resemblance to the present model. In that work, "object spaces," a form of comonoids of coherent spaces, were used for modelling stores. This was the first instance of sophisticated mathematical objects being used to model stores and provided inspiration for other models such as the one proposed here. Beyond this, it is hard to draw any firm conclusions about a correspondence between the two because the model of [24] is event-based and stateless, whereas representing states is an important objective of the present model.

In recent work, Ahmed, Dreyer and colleagues [3, 10] have applied the ideas of possible worlds (similar to functor categories) and automata-theoretic reasoning in the setting of operational reasoning. While the ideas seem intuitively similar, it is difficult to make a formal comparison at the present stage because the starting points of denotational and operational approaches are quite different. Some remarks regarding the comparison may be found in [34]. It is also worth remarking that these researchers have not yet tackled the issues of passivity in their approaches.

In another line of work, Benton et al. [5] have proposed a semantic characterization of effect systems in a global store model using relation-preservation properties. They were led to analyze "observable read-only effects" (i.e., observable passivity) as well as its dual "observable write-only effects," and their characterization turns out to be quite similar to ours, *viz.*, passive computations are those that preserve the identity relations on states. These ideas have been extended to dynamic allocation of stores using Kripke logical relations (similar to our functor category models) in subsequent work [4, 42]. The key difference between their work and ours is that they model effect systems, which may be thought of as Curry-style properties of computations, whereas we model type systems in the Church-style using semantic structures. The delicate balance of intensional and extensional effects does not seem to arise in this line of work.

7 Conclusion

We have defined a conceptually-based semantic model for imperative programs that captures the notion of "passivity". This is done using a recently developed automata-theoretic denotational framework, where stores are modelled as an abstract form of automata, with explicit representation of states as well as state transitions. Relational parametricity of the type and term interpretations then ensures that the properties of passive expressions are respected.

This approach contrasts with the intensional models such as the event-based and games models [2, 32] where passivity is modelled by "fiat," by designating certain events or moves as passive ones. While such models have strong definability and full abstraction properties, they however lack an explanation of what it means for a computation to be passive. In our extensional framework, on the other hand, a computation is passive if it is *independent* of the state transformations that might be possible in the store. We believe this gives a clear answer to the semantic question of what passivity means.

One might wonder if the model presented here is fully abstract. We have not investigated the question in detail and it will perhaps involve considerable work to settle the question because functor categories are quite extensive and not enough is not known about what is definable in them. However, we are able to calculate explicit representation results for simple first order types such as $\textsc{Com} \Rightarrow \textsc{Com}$ and $\textsc{Com} \Rightarrow \textsc{Exp}$, which are accurate. We leave a full exploration of the full abstraction question to future work.

Other questions that this work might enable is a semantic understanding of the various notions of passivity present in specification and verification frameworks, e.g., program specification systems [19], ownership type systems [22] and fractional permission-based methods [7, 35]. Secondly, the successful modeling of passivity takes us one step closer to modeling program logics such as Syntactic Control of Interference [23], Specification Logic [37, 41] and Separation Logic [35, 39]. We envisage that the model presented here will be helpful to streamline the semantic treatment of such programming logics.

Acknowledgements

We are grateful to anonymous referees whose comments led to various improvements in the paper. Discussions with Claudio Hermida, Neil Ghani and Patricia Johann are warmly acknowledged.

References

[1] S. Abramsky and G. McCusker. Linearity, sharing and state. In *Linear Logic 96 Tokyo Meeting*, volume 3 of *Elect. Notes in Theor. Comput. Sci.*, pages 2–14. Elsevier, 1996. (Also as Chapter 20 of [27]).

[2] S. Abramsky and G. McCusker. Full abstraction for Idealized Algol with passive expressions. *Theoretical Comput. Sci.*, 227(1-2):3–42, 1999.

[3] A. Ahmed, D. Dreyer, and A. Rossberg. State-dependent representation independence. In *Thirty Sixth Ann. ACM Symp. on Princ. of Program. Lang.* ACM, 2009.

[4] N. Benton, A. Kennedy, L. Beringer, and M. Hofmann. Relational semantics for effect-based program transformations with dynamic allocation. In *9th ACM SIGPLAN Intl. Conf. on Princ. and Practice of Declarative Program.*, pages 87–96. ACM, 2007.

[5] N. Benton, A. Kennedy, M. Hofmann, and L. Beringer. Reading, writing and relations. In N. Kobayashi, editor, *APLAS '06*, pages 114–130. Springer, 2006.

[6] R. Bornat, C. Calcagno, P.W. O'Hearn, and M. Parkinson. Permission accounting in Separation Logic. In *ACM Symp. on Princ. of Program. Lang.*, pages 59–70. ACM Press, 2005.

[7] J. Boyland. Checking interference with fractional permissions. In R. Cousot, editor, *Static Analysis: 10th Intern. Symp.*, volume 2694 of *LNCS*, pages 55–72. Springer, 2003.

[8] S. D. Brookes. A semantics for Concurrent Separation Logic. *Theoretical Comput. Sci.*, 375(1-3):227–270, Apr 2007.

[9] S. D. Brookes, M. Main, A. Melton, and M. Mislove, editors. *Math. Found. of Programm. Semantics: Eleventh Ann. Conference*, volume 1 of *Elect. Notes in Theor. Comput. Sci.* Elsevier, 1995.

[10] D. Dreyer, G. Neis, and L. Birkedal. The impact of higher-order state and control effects on local relational reasoning. In *ICFP*, 2010.

[11] B. P. Dunphy and U. S. Reddy. Parametric limits. In *Proc. 19th Ann. IEEE Symp. on Logic in Comp. Sci.*, pages 242–253. IEEE, July 2004.

[12] S. Eilenberg. *Automata, Languages, and Machines.* Academic Press, 1974. (Volumes A and B).

[13] P. J. Freyd, P. W. O'Hearn, A. J. Power, M. Takeyama, and R. D. Tennent. Bireflectivity. In Brookes et al. [9], pages 199–213.

[14] C. Hermida. Fibrations, logical predicates and indeterminantes. Ph.D. thesis and Technical Report ECS-LFCS-93-277, University of Edinburgh, 1993.

[15] C. A. R. Hoare. An axiomatic basis for computer programming. *Comm. ACM*, 12:576–583, 1969.

[16] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, London, 1985.

[17] W. M. L. Holcombe. *Algebraic Automata Theory.* Cambridge Studies in Advanced Mathematics. Cambridge Univ. Press, Cambridge, 1982.

[18] B. Jacobs. *Categorical Logic and Type Theory*, volume 141 of *Studies in Logic and the Foundations of Mathematics.* Elsevier, 1999.

[19] K. R. M. Leino. Dafny: An automatic program verifier for functional correctness. In *LPAR-16*, volume 6355 of *LNCS*, pages 348–370. Springer-Verlag, 2010.

[20] G. McCusker. A fully abstract relational model of syntactic control of interference. In *Computer Science Logic (CSL) 2002*, volume 2471 of *LNCS*, pages 247–261. Springer-Verlag, 2002.

[21] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[22] J. Noble, J. Vitek, and J. Potter. Flexible alias protection. In E. Jul, editor, *ECOOP'98 - Object-oriented Programming*, volume 1445 of *LNCS*, pages 158–185. Springer-Verlag, 1988.

[23] P. W. O'Hearn, A. J. Power, M. Takeyama, and R. D. Tennent. Syntactic control of interference revisited. In Brookes et al. [9], pages 447–486. (Reprinted as Chapter 18 of [27]).

[24] P. W. O'Hearn and U. S. Reddy. Objects, interference and the Yoneda embedding. *Theoretical Computer Science*, 228(1):211–252, 1999.

[25] P. W. O'Hearn and J. C. Reynolds. From Algol to polymorphic linear lambda-calculus. *J. ACM*, 47(1):167–223, Jan 2000.

[26] P. W. O'Hearn and R. D. Tennent. Parametricity and local variables. *J. ACM*, 42(3):658–709, 1995. (Reprinted as Chapter 16 of [27]).

[27] P. W. O'Hearn and R. D. Tennent. *Algol-like Languages (Two volumes)*. Birkhäuser, Boston, 1997.

[28] F. J. Oles. *A Category-Theoretic Approach to the Semantics of Programming Languages.* PhD thesis, Syracuse University, 1982.

[29] F. J. Oles. Functor categories and store shapes. In *Algol-like Languages* [27], chapter 11, pages 3–12.

[30] G. Plotkin and M. Abadi. A logic for parametric polymorphism. In *Typed Lambda Calculi and Applications - TLCA '93*, LNCS, pages 361–375. Springer-Verlag, 1993.

[31] U. S. Reddy. Passivity and independence. In *Proc. Ninth Ann. IEEE Symp. on Logic in Comp. Sci.*, pages 342–352. IEEE, July 1994.

[32] U. S. Reddy. Global state considered unnecessary: An introduction to object-based semantics. *J. Lisp and Symbolic Computation*, 9:7–76, 1996. (Reprinted as Chapter 19 of [27]).

[33] U. S. Reddy and B. P. Dunphy. An automata-theoretic model of objects. In E. Zucca, editor, *2011 Intl. Workshop on Foundations of Object-Oriented Languages*, pages 1–15. electronic proceedings at http://www.disi.unige.it/person/ZuccaE/FOOL2011/, 2011.

[34] U. S. Reddy and B. P. Dunphy. An automata-theoretic model of Idealized Algol. In *Automata, Languages and Programming (ICALP 2012)*, volume 7392 of *LNCS*, pages 337–350. Springer-Verlag, 2012.

[35] U. S. Reddy and J. C. Reynolds. Syntactic control of interference for Separation Logic. In *Thirty Ninth Ann. ACM Symp. on Princ. of Program. Lang.*, pages 323–336. ACM, 2012. (ACM SIGPLAN Notices, 47:1:323-336).

[36] J. C. Reynolds. The essence of Algol. In J. W. de Bakker and J. C. van Vliet, editors, *Algorithmic Languages*, pages 345–372. North-Holland, 1981. (Reprinted as Chapter 3 of [27]).

[37] J. C. Reynolds. Idealized Algol and its specification logic. In D. Neel, editor, *Tools and Notions for Program Construction*, pages 121–161. Cambridge Univ. Press, 1982. (Reprinted as Chapter 6 of [27]).

[38] J. C. Reynolds. Types, abstraction and parametric polymorphism. In R. E. A. Mason, editor, *Information Processing '83*, pages 513–523. North-Holland, Amsterdam, 1983.

[39] J.C. Reynolds. Separation Logic: A logic for shared mutable data structures. In *LICS*, pages 55–74, 2002.

[40] D. S. Scott and C. Strachey. Towards a mathematical semantics for computer languages. In J. Fox, editor, *Proc. of Symp. on Computers and Automata*, pages 19–46. Polytech Institute of Brooklyn Press, 1971. (original Tech. Report Oxford PRG-6.).

[41] R. D. Tennent. Semantical analysis of specification logic. *Inf. Comput.*, 85(2):135–162, 1990. (Reprinted as Chapter 13 of [27]).

[42] J. Thamsborg and L. Birkedal. A Kripke logical relation for effect-based program transformations. *SIGPLAN Not.*, 46(9):445–456, September 2011.

[43] P. Wadler. Linear types can change the world! In M. Broy and C. B. Jones, editors, *Program. Concepts and Methods*. North-Holland, Amsterdam, 1990. (Proc. IFIP TC 2 Working Conf., Sea of Galilee, Israel).

Appendix

Definitions

In this section, we give a brief overview of the framework of reflexive graphs [26, Sec. 7] and parametricity graphs [11].

Formally, we are considering reflexive graph objects in **CAT**, the category of all (small) categories.

Unpacking the definition, we note that a reflexive graph $\mathbf{G}$ consists of two categories $\mathbf{G}_v$ and $\mathbf{G}_e$ (the "vertex" category and the "edge" category, respectively), and three functors between them $\partial_9, \partial_1 : \mathbf{G}_e \to \mathbf{G}_v$ and $I : \mathbf{G}_v \to \mathbf{G}_e$ such that $\partial_i \circ I = \mathrm{Id}_{\mathbf{G}_v}$. The functors ∂_0 and ∂_1 pick out the "source" and the "target" for the edges and their morphisms, whereas I assigns to each vertex X an "identity" edge I_X. The notation $R : X \leftrightarrow X'$ is used to denote the situation that $\partial_0(R) = X$ and $\partial_1(R) = X'$. The definition also generalize to edges of arbitrary arity in place of binary edges.

Reflexive graphs represent a special case of indexed categories. Hence, they form a 2-category with 1-cells being called "RG-functors" and 2-cells being called "parametric natural transformations".

Intuitively, this data means that we use two-dimensional categorical structures, where morphisms occupy one dimension and edges (modelling "relations") between categorical objects occupy the second dimension, as in the diagram below:

$$\begin{tikzcd}
X \arrow[r, "f"] \arrow[d, leftrightarrow, "R"'] & Y \arrow[d, leftrightarrow, "S"] \\
X' \arrow[r, "f'"] & Y'
\end{tikzcd}$$

A diagram of this form, called a *square*, is the shape of a morphism in $\mathbf{G}_e$ (of type $R \to S$ with its "source" and "target" being f and f'). It represents the property that the morphisms f and f' map R-related arguments to S-related results. The textual notation for the property is $f\ \big[R \to S\big]\ f'$.

A reflexive graph is called *relational* if there is at most one edge morphism of any given shape. In that case, the hom-set $\mathbf{G}_e[R, S]$ is a set-theoretic relation between $\mathbf{G}_v[X, Y]$ and $\mathbf{G}_v[X', Y']$.

The reflexive graphs we work with are called *parametricity graphs* [11]. They incorporate additional axioms to capture the idea that relations in the vertical dimension indeed behave like "relations" in the intuitive sense. A parametricity graph is a reflexive graph that (i) is relational (ii) satisfies the *identity condition*: $f \left[I_X \to I_Y\right] f' \Longrightarrow f = f'$ and (iii) has a cloven fibration $\langle \partial_0, \partial_1 \rangle : \mathbf{G}_e \to \mathbf{G}_v \times \mathbf{G}_v$. The last of these conditions, which is an established part of category theory [18], means the following. The right square $f \left[R \to S\right] f'$ in the diagram below is called a *cartesian square* if every square of the form of the outer square uniquely factors through it:

$$\begin{tikzcd}
X \arrow[r, "g"] \arrow[d, leftrightarrow, "P"'] & A \arrow[r, "f"] \arrow[d, leftrightarrow, dotted, "R"'] & B \arrow[d, leftrightarrow, "S"] \\
X \arrow[r, "g'"] & A' \arrow[r, "f'"] & B'
\end{tikzcd}$$

The reflexive graph is *fibred* if, for all f, f' and S of matching types, there is an edge R that fills the dotted arrow making it a cartesian square. The edge R is unique up to isomorphism. A particular choice of such edges $\langle f, f' \rangle^* \, S = R$ is called a *cleavage* and the fibration is said to be *cloven*. Parametricity graphs are given with a chosen cleavage (even though in most of our examples, the cleavage is unique).

A parametricity graph-functor (PG-functor) is an RG-functor that preserves the chosen cleavage. A 2-cell between such functors (a parametric natural transformation) only needs to satisfy the parametricity condition; naturality follows from parametricity [11]. This is because parametricity graphs have a *subsumption map* $\langle - \rangle$ that sends morphisms $g : X \to X'$ to edges $\langle g \rangle : X \leftrightarrow X'$ with the property that a square of shape on the left below exists iff the square of morphisms on the right commutes:

$$\begin{tikzcd}
X \arrow[r, "f"] \arrow[d, leftrightarrow, "\langle g \rangle"'] & Y \arrow[d, leftrightarrow, "\langle h \rangle"] & & X \arrow[r, "f"] \arrow[d, "g"'] & Y \arrow[d, "h"] \\
X' \arrow[r, "f'"] & Y' \arrow[r, phantom, "\Longleftrightarrow", start anchor=north east, end anchor=north west] & & X' \arrow[r, "f'"] & Y'
\end{tikzcd}$$

The subsumption map is given by $\langle g \rangle = \langle g, \mathrm{id}_{X'} \rangle^* \, I_{X'}$.

Dually, *co-cartesian squares* are of the form of the left inner square in the diagram:

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, leftrightarrow, "R"'] & B \arrow[r, "g"] \arrow[d, leftrightarrow, dotted, "S"] & X \arrow[d, leftrightarrow, "T"] \\
A \arrow[r, "f'"] & B' \arrow[r, "g'"] & X'
\end{tikzcd}$$

so that all outer squares factor through them. An RG-functor is *cofibred* if it maps all co-cartesian squares that exist in its source graph to co-cartesian squares in the target graph. We make use of PG-functors that are cofibred. However, we do not require that the source graph itself should be cofibred, i.e., not all R, f and f' are required to have corresponding S relations.

Linearization of Automatic Arrays and Weave Specifications

David Sprunger[1]

Department of Mathematics
Indiana University
Bloomington, Indiana

Abstract

Grabmayer, Endrullis, Hendriks, Klop, and Moss [6] developed a method for defining automatic sequences in terms of 'zip specifications' and proved that a sequence is automatic [2] iff it has a zip specification where all zip terms have the same arity. An open question at the end of that work regards what kinds of sequences are given by zip specifications where the zip terms do *not* have a constant arity and whether it can be decided if two such specifications have the same solution.

This paper begins by investigating a similar definitional scheme for the higher-dimensional counterpart of automatic sequences, automatic arrays. In the course of establishing the results required for this machinery, we find an isomorphism between a final coalgebra for arrays and the standard final coalgebra for sequences which is closely related to the z-order curve [8]. This isomorphism preserves automaticity properties: an array is k, l-automatic iff its corresponding sequence is kl-automatic. The former notion of automaticity (k, l-automatic, note the comma) is defined for arrays as in [2], and the latter notion is the standard notion of automaticity for sequences. It also provides a convenient way to translate between stream zip specifications and array zip specifications. Finally, we provide an example application of automatic arrays to provide a partial answer (in the affirmative) to the question of decidability of mixed zip specifications. If a zip specification with zip terms of different arities satisfies a particular condition, we will create an automaton to generate the solution to this zip-specification as an automatic array. Then we will use our earlier work to check whether two arrays generated for two of these special zip-mix specifications are equal.

Keywords: automatic sequence, automatic array, coalgebra, final coalgebra, z-order curve, zip specification

1 Introduction

The automatic sequences are a class of sequences arising naturally in both computer science and many different fields of mathematics [2]. A well-known automatic sequence, the Prouhet-Thue-Morse sequence, has been rediscovered over and over again by its appearances in combinatorics, algebra, number theory, differential geometry, and combinatorial game theory. [2]

As an example, sequences are interesting to algebraists as the expansions of numbers in particular bases. A number is rational, for example, iff its represen-

[1] Email: dasprung@indiana.edu

[2] See [1] for more details on the appearances of the PTM sequence.

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.com/locate/entcs

tation in base k is an eventually periodic sequence of digits. Sequences which are eventually periodic are highly regular; at the other end of the spectrum are the totally random sequences, the investigation of which relies primarily on probabilistic methods and/or information-theoretic methods. Automatic sequences form a class of sequences "more random" than the eventually periodic sequences and yet with enough structure that their properties can be investigated without needing probabilistic methods. Numbers whose base k expansion are automatic sequences are useful concrete examples in the study of transcendental numbers. Automatic sequences also play a crucial role in determining transcendence of power series over function fields—this is the content of Christol's theorem [3].

While the identities and properties of particular automatic sequences are often encountered and used in mathematics, their definition and the method by which they are generated lies more naturally in the realm of theoretical computer science. Automatic sequences list the outputs of a simple class of automata on representations of the natural numbers. This class of automata is a slight generalization of the better known deterministic finite automata (DFA), which is the preferred model in formal language theory for recognizing the regular languages. As a consequence of this similarity, the methodologies used for regular languages and automatic sequences have strong resemblances to each other, but also have some important differences.

Regular languages enjoy a well developed notational system for defining languages and language operations, namely regular expressions. Commonly juxtaposition or $\cdot$ indicate language concatenation, L^* denotes the Kleene closure of L, and $\cup$ or $+$ commonly indicates the union of languages. Since the words in regular languages are finite, this definitional scheme is able to compactly describe languages while simultaneously hinting at the structure of their elements.

On the other hand, the class of automatic sequences has necessarily infinite objects, so schemes giving a finite definition of even one element of this space require some thought. Given two finite presentations of infinite objects like sequences, it can be quite difficult to determine whether the two generated objects are even equal. To reason about infinite objects, principles of coinduction and corecursion arising from coalgebra are often employed, as introduced by Rutten [10]. Though the coalgebraic approach is not the original automata theory approach to regular languages, the theory of regular languages and DFAs has been recast in those terms by Rutten [9].

The standard finite scheme to define an automatic sequence is to give a finite automaton which generates that sequence. This scheme is useful when computing entries in the sequence, but suffers from some ambiguities involving input representation. [6] proposed an alternate definitional scheme which gives a sequence as the result interleaving the entries of other sequences. This work showed the collection of automatic sequences exactly coincides with the sequences that can be defined with zip specifications.

In this paper we first by summarize the main techniques used in [6] to define zip-specifications for automatic sequences. Then we begin to investigate a similar scheme for automatic arrays, which are a two-dimensional generalization of automatic sequences. This investigation will uncover an interesting isomorphism between automatic arrays and sequences, which we will go on to use to give a partial answer to an open question regarding zip specifications.

2 A brief introduction to automatic sequences, coalgebras, and zip specifications

We begin by outlining the primary techniques used in the study of automatic sequences and coalgebras. For readers seeking more details on the former topic there is a standard book by Allouche and Shallit [2]. For the latter, there are many papers by Rutten and many others expositing coalgebraic methods most notably [10], see also [7] particularly for coalgebraic methods applied to sequences of symbols. With some understanding of these techniques, we then turn to how these techniques are used to construct zip-specifications and check for equality of solutions.

2.1 Automata with output and automatic sequences

We now describe in greater detail the process of generating a sequence from an automaton [2,4]. A **DFAO** (**D**eterministic **F**inite **A**utomaton with **O**utput) is an automaton with the following components: an input alphabet (A), a finite set of states (Q), a designated start state ($q_0 \in Q$), a transition map taking the current state and an input symbol and returning the next state ($\delta : Q \times A \to Q$), an output alphabet ($\Delta$) and an output map ($f : Q \to \Delta$). These generalize their better known cousin, the DFA, in that a DFA's output alphabet is necessarily Δ = {accept, reject}.

We can extend a DFAO's transition function from single letters in A to all words in A^* with the following recursive scheme:

$$\delta(q, w) = \begin{cases} q & \text{if } w = \epsilon \\ \delta(\delta(q, a), w') & \text{if } w = aw' \end{cases}$$

for $w, w' \in A^*$, $a \in A$ and ϵ being the empty word in A^*. As we are about to note, this is not the only possible nor the only common way to extend the transition function to all of A^*. We will indicate that a DFAO uses *this* convention for its extended transition function by saying the DFAO "reads its input in left-to-right order" or "uses the frontwards order".

We could alternatively extend a DFAO's transition function to all of A^* with a slightly different recursive scheme:

$$\delta(q, w) = \begin{cases} q & \text{if } w = \epsilon \\ \delta(\delta(q, a), w') & \text{if } w = w'a \end{cases}$$

In this case, we say the DFAO reads its input in the right-to-left order or the backwards order.

In either case, we say the DFAO outputs $d \in \Delta$ on input $w \in A^*$ if $d = f(\delta(q_0, w))$. We say a DFAO is a k-DFAO if $|A| = k \in \omega$ and in this case assume

$A = \{0, 1, 2, \ldots, k-1\} = \mathbb{N}_{<k}$. Note that each integer n is naturally associated to a string in A^*: its standard base-k representation which we denote by $[n]_k \in \mathbb{N}^*_{<k}$. We say a k-DFAO generates the sequence $\sigma = \sigma_0\sigma_1\sigma_2 \ldots \sigma_n \ldots$ if $\sigma_n = f(\delta(q_0, [n]_k))$. That is, the DFAO outputs σ_n on the input $[n]_k$.[3] If σ is a sequence generated by a k-DFAO, we say it is k-automatic.

At this point, there are a few good and well-answered questions to point out. How is the collection of sequences generated by k-DFAOs using the frontwards order related to the collection of sequences generated by k-DFAOs using the backwards order? Is it always possible to modify a k-DFAO to allow the representation $[n]_k$ to have leading 0's without affecting the stream it generates? It turns out the collection of automatic sequences is exactly the same under each of these input conventions, and indeed the functions taking a DFAO with one input convention to the others are computable [2, p. 159].

Though it is more common in the general literature on automatic sequences to encounter the frontwards order input convention, when using coalgebraic techniques there are some reasons to prefer the backwards order convention. As a result, we will assume for the rest of the paper that all our automata read their input in the backwards order.

2.2 Coalgebras, finality and bisimulation

The next critical component in this machinery is the notion of coalgebras for a functor [10]. In this paper we will be considering the Set endofunctor $F : X \mapsto \Delta \times X^k$ where the functor acts on arrows by sending the Set morphism $f : X \to Y$ to $Ff : \Delta \times X^k \to \Delta \times Y^k$ defined by $Ff = \langle id_\Delta, f, f, \ldots, f\rangle$. A **coalgebra for this functor** is a set C together with a function $c : C \to FC = \Delta \times C^k$.

The map c is usually called the structure map for the coalgebra, while C is its carrier. We will often need to talk about the different components of the structure map, $c = \langle o, c_0, c_1, \ldots, c_{k-1}\rangle$, where $o : C \to \Delta$ and $c_i : C \to C$. o is often called the **observation component** of the structure, while the c_i are called the **transitions** for the structure.

We define the category of F-coalgebras as follows: objects in the category are F-coalgebras like (C, c) and a morphism of coalgebras $\varphi : (C, c) \to (D, d)$ is a function $\varphi : C \to D$ such that the following diagram commutes:

$$\begin{tikzcd}
C \arrow[r, "c"] \arrow[d, "\varphi"'] & \Delta \times C^k \arrow[d, "F\varphi"] \\
D \arrow[r, "d"'] & \Delta \times D^k
\end{tikzcd}$$

We say an F-coalgebra is **final** if it is a final object in the category of F-coalgebras. Final coalgebras are particularly important in the theory of coalgebras because there is a powerful technique for establishing the equality of two elements of a final coalgebra: bisimulation.

A **bisimulation** on an F-coalgebra (C, c) is a relation $R \subseteq C \times C$ such that for

[3] Note the DFAO is not a Mealy/Moore machine or other kind of simple transducer, so it is not generating this sequence in a single run. Each run of the DFAO produces a single output and by listing these individual results of each of the runs for these particular inputs we get a sequence.

all $(x, y) \in R$ we have $o(x) = o(y)$ and for all $0 \leq i < k$ we have $(c_i(x), c_i(y)) \in R$. A standard theorem of coalgebra is that every bisimulation on a final coalgebra is a subset of the identity relation [10,11]. That is, if R is a bisimulation on a final coalgebra and $(x, y) \in R$, then $x = y$.

2.3 DFAOs and the set of sequences as coalgebras

Coalgebras and bisimulations are commonly used in the analysis of state transition systems and other process calculi. DFAs and DFAOs are simple examples of state transition systems and so they form a natural example of coalgebras and thereby a place to do bisimulations. Less readily apparent is the fact that there is also a coalgebra structure on the set of sequences themselves. [4]

A k-DFAO generating an automatic sequence naturally gives an F-coalgebra via the structure map $Q \to \Delta \times Q^k$ given by $\langle f, \delta(\cdot, 0), \delta(\cdot, 1), \ldots, \delta(\cdot, k-1)\rangle$. As a result of the so-called input robustness results mentioned in section 2.1, we can also assume without loss of generality that our DFAOs (which read in the backwards order) ignore leading zeroes. That is, we have $f \circ \delta(\cdot, 0) = f$. This property, as will be discussed later, is called zero-consistency.

An important object in the study of automatic sequences is the so-called kernel of a sequence. To define this, we first define projection maps on the set of sequences in Δ. The projection $\pi_{i,k} : \Delta^\omega \to \Delta^\omega$ is given by $\pi_{i,k}(\sigma) = \sigma_i \sigma_{i+k} \sigma_{i+2k} \ldots \sigma_{i+nk} \ldots = \{\sigma_{i+nk}\}_n$. The k-kernel of the sequence σ is the set of all sequences which can be reached by repeatedly applying the maps $\pi_{0,k}, \pi_{1,k}, \ldots, \pi_{k-1,k}$ to σ. A well-known theorem states that a sequence is k-automatic iff its k-kernel is finite [2, p. 185].

Now, it is a fact due to [7] and [6] independently that Δ^ω is the carrier for a coalgebra $(\Delta^\omega, \langle \mathtt{hd}, \pi_{0,k}, \pi_{1,k}, \ldots, \pi_{k-1,k}\rangle)$ which is final for zero-consistent F-coalgebras, which form a subcategory of the F-coalgebras. Therefore there is a unique map, seq, such that the following diagram commutes:

$$\begin{tikzcd}
Q \arrow[r, "{\langle f,\delta(\cdot,0),\delta(\cdot,1),\ldots,\delta(\cdot,k-1)\rangle}"] \arrow[d, "\mathrm{seq}"'] & \Delta \times Q^k \arrow[d, "F\,\mathrm{seq}"] \\
\Delta^\omega \arrow[r, "{\langle \mathtt{hd},\pi_{0,k},\pi_{1,k},\ldots,\pi_{k-1,k}\rangle}"'] & \Delta \times (\Delta^\omega)^k
\end{tikzcd}$$

The seq map takes a state q to the automatic sequence generated by the DFAO when using q as the start state.

2.4 The $\mathbf{zip}_k$ function and zip-specifications

Now that we understand automatic sequences as coalgebras and can decide when two elements of a final coalgebra are equal, we are ready to discuss zip-specifications, as treated in [6]. The $\mathbf{zip}_k$ function takes k sequences and merges them into a single sequence by alternating through their terms. For example, $\mathbf{zip}_2(\sigma, \tau) = \mathbf{zip}_2(\sigma_0\sigma_1\sigma_2\ldots, \tau_0\tau_1\tau_2\ldots) = \sigma_0\tau_0\sigma_1\tau_1\sigma_2\tau_2\ldots$. The $\mathbf{zip}_k$ function in some sense inverts the action of the k-kernel maps: $\pi_{i,k}(\mathbf{zip}_k(\sigma_0, \sigma_1, \ldots, \sigma_{k-1})) = \sigma_i$ and conversely $\mathbf{zip}_k(\pi_{0,k}(\sigma), \pi_{1,k}(\sigma), \ldots, \pi_{k-1,k}(\sigma)) = \sigma$.

[4] For more details on the coalgebraic structure of DFAs and their final coalgebra, see [9].

Using the **zip** function, we can set up systems of equations to define streams. For example, the system

$$\begin{cases} m = 0 : x \\ x = 1 : \mathbf{zip}(x, y) \\ y = 0 : \mathbf{zip}(y, x) \end{cases} \tag{1}$$

has the Thue-Morse sequence as the solution for m.

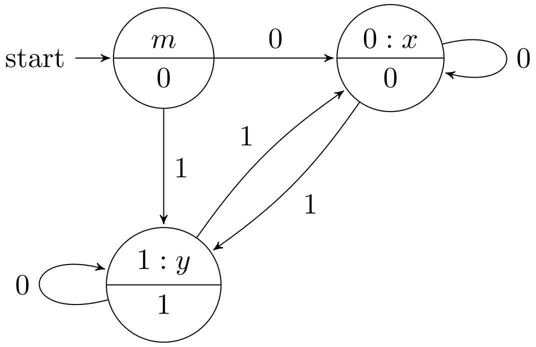
To be exact, a **zip specification** in an alphabet A is a set of variables, S, along with a zip term for each variable, where a zip term is generated by the BNF grammar

$$T ::= s \mid a : T \mid \mathbf{zip}_k(T, T, \dots, T)$$

where $s \in S$, $a \in A$, and k zip terms are provided as the argument for a term using the $\mathbf{zip}_k$ rule. For example, $\mathbf{zip}_2(1 : x, \mathbf{zip}_3(0 : y, x, 1 : 0 : z))$ is a zip term in the alphabet $\{0, 1\}$ with variables $\{x, y, z\}$. A zip specification gives a zip term for each variable in the set.

Grabmayer et al. [6] gave conditions for the existence and uniqueness of solutions to these zip specifications. Roughly stated, there is a solution to the specification if each variable is defined by exactly one zip term, and there is a unique solution if for each variable the first symbol of that variable's term can be determined. They then proved that solutions to zip specifications where all terms using the **zip** function have arity k are k-automatic. This tells us that all solutions for the variables in specification (1) are 2-automatic, for example.

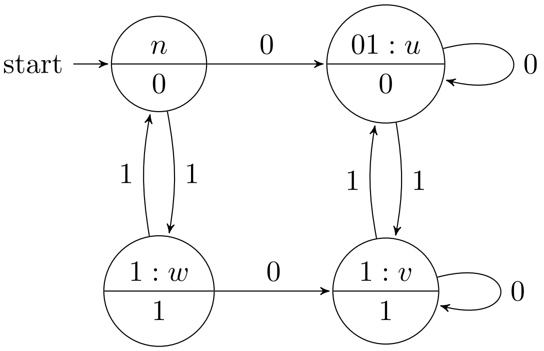
Generating an automaton for (1) is fairly straightforward. We start with the root (first) variable and apply the $\pi_{i,2}$ maps just as if we're generating the 2-kernel of a sequence, but we get state names instead. For example, $\pi_{0,2}(m) = \pi_{0,2}(0 : x) = \pi_{0,2}(01 : \mathbf{zip}(x, y)) = 0 : \pi_{0,2}(\mathbf{zip}(x, y)) = 0 : x$, so after reading 0, our automaton will transition from a state labelled m to a state labelled $0 : x$. Similarly, $\pi_{1,2}(0 : x) = \pi_{1,2}(01 : \mathbf{zip}(x, y)) = 1 : y$ so the 1 transition out of $0 : x$ goes to $1 : y$ and so on. The full 2-DFAO for this specification is given below:



Now given a second zip specification, such as

$$\begin{cases} n = 0 : \mathbf{zip}(1 : w, 1 : u) \\ u = 1 : \mathbf{zip}(v, u) \\ v = 0 : \mathbf{zip}(v, 1 : u) \\ w = \mathbf{zip}(n, v) \end{cases} \tag{2}$$

it is fairly straightforward to check that the first few entries in the solution for n match the first few entries for the solution of m, but is a nearly impossible game of index bookkeeping to check that they have the same values at all entries. To aid our understanding and to simplify matters we use the machinery of coalgebras. First we generate a 2-DFAO for this specification in the same manner as the one before it:



Next we find attempt to find a bisimulation from the first DFAO to the second relating m to n. $R = \{(m, n), (0 : x, 01 : u), (0 : x, n), (1 : y, 1 : w), (1 : y, 1 : v)\}$ is such a bisimulation. Then if we apply the seq map to this bisimulation it remains a bisimulation in Δ^ω. In particular, $(\mathrm{seq}(m), \mathrm{seq}(n)) \in \mathrm{seq}(R)$, which is a bisimulation in a final coalgebra, so $\mathrm{seq}(m) = \mathrm{seq}(n)$. This allows us to conclude the sequence generated in the first DFAO starting from m is the same as the sequence generated from the second DFAO starting from state n.

2.5 Automatic arrays

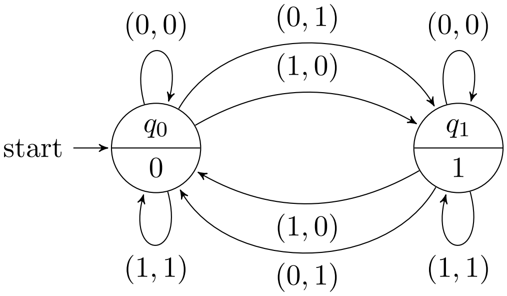
Automatic arrays are a generalization of automatic sequences to two dimensions. [2, Ch. 14]

Let's imagine we have a DFAO with input alphabet $\mathbb{N}_{<k} \times \mathbb{N}_{<l}$. (We will sometimes abbreviate this situation with the term k, l-DFAO.) We can encode a pair $(m, n) \in \mathbb{N} \times \mathbb{N}$ as a word in this alphabet in a standard way: we find the base-k representation of m, $[m]_k$, and the base-l representation of n, $[n]_l$, and pad whichever has fewer digits with leading zeroes until they are the same length. Then the standard k, l-encoding of (m, n) is the sequence of pairs of these digits starting with the most significant digits.

For example, the standard 2,5-encoding for $(13, 82)$ is formed by taking $[13]_2 = 1101$ and $[82]_5 = 312 = 0312$ and then forming pairs: $(1, 0)(1, 3)(0, 1)(1, 2)$.

Now imagine a quarter infinite grid with positions indexed by $\mathbb{N} \times \mathbb{N}$ with the symbol at position (m, n) being the output of the DFAO when given the standard encoding for (m, n) in $\mathbb{N}_{<k} \times \mathbb{N}_{<l}$. Such an infinite array is called a **k, l-automatic array** in analogy to the definition of automatic sequences. When the bases k and l can be inferred from context, we may abbreviate and use the term "automatic array".

As an example consider the following automaton and the automatic array it generates below.[5] Note that this automaton has input alphabet $\mathbb{N}_{<2} \times \mathbb{N}_{<2}$.



$$\begin{array}{|cccccccccccccccc}
0&1&1&0&1&0&0&1&1&0&0&1&0&1&1&0\\
1&0&0&1&0&1&1&0&0&1&1&0&1&0&0&1\\
1&0&0&1&0&1&1&0&\mathbf{0}&1&1&0&1&0&0&1\\
0&1&1&0&1&0&0&1&1&0&0&1&0&1&1&0\\
1&0&0&1&0&1&1&0&0&1&1&0&1&0&0&1\\
0&1&1&0&1&0&0&1&1&0&0&1&0&1&1&0\\
0&1&1&0&1&0&0&1&1&0&0&1&0&1&1&0\\
1&0&0&1&0&1&1&0&0&1&1&0&1&0&0&1\\
1&0&0&1&0&1&1&0&0&1&1&0&1&0&0&1\\
0&1&1&0&1&0&0&1&1&0&0&1&0&1&1&0\\
0&1&1&0&1&0&0&1&1&0&0&1&0&1&1&0\\
1&0&0&1&0&1&1&0&0&1&1&0&1&0&0&1\\
0&1&1&0&1&0&0&1&1&0&0&1&0&1&1&0\\
1&0&0&1&0&1&1&0&0&1&1&0&1&0&0&1\\
1&0&0&1&0&1&1&0&0&1&1&0&1&0&0&1\\
0&1&1&0&1&0&0&1&1&0&0&1&0&1&1&0\\
\hline
\end{array}$$

The bold zero on this line is at (13, 8). This gets encoded as (1,1)(1,0)(0,0)(1,0). You can check to see that this is the right output using the automaton.

2.6 Previously known results on automatic arrays

At this point the same kinds of questions that are natural for automatic sequences can be asked for automatic arrays. Does the collection of automatic arrays change if we make our k, l-DFAOs read in backwards order vs. frontwards order? Can we allow leading $(0, 0)$ characters for these DFAOs without changing the notion of which arrays are k, l-automatic? These standard input robustness results work out the same way for automatic arrays as they do for sequences: changing the direction of the input and adding leading zeroes does not change the collection of automatic arrays [2, p. 408].

Indeed, very many of the results for automatic sequences have a corresponding result for automatic arrays. One important example of this phenomenon regards the kernel of an array. We define $\pi_{\frac{i}{k}, \frac{j}{l}}$ to take an array a with entries $a_{m,n}$ for all

[5] This automaton is intentionally similar to the minimal automaton that generates the Thue-Morse sequence. The array it generates is also intimately related to the Thue-Morse sequence. Every row and every column is the Thue-Morse sequence or the Thue-Morse sequence with 0 and 1 swapped. The array's 2,2-linearization is also the Thue-Morse sequence. See 3.2.

$n, m \geq 0$ to the array with entries $a_{km+i,ln+j}$ for all $n, m \geq 0$. Then the k, l-kernel of the array a is the set of all arrays which can be reached from a by repeatedly applying the functions $\pi_{\frac{i}{k},\frac{j}{l}}$ for $i \in \mathbb{N}_{<k}$ and $j \in \mathbb{N}_{<l}$. It turns out the k, l-kernel of an array is finite iff the array is k, l-automatic [2, p. 409].

3 Weave specifications for automatic arrays

To establish results for zip specifications for sequences, several steps were needed. First, one must give conditions for existence and uniqueness of solutions to zip specifications. Then, for zip specifications fulfilling these criteria, one shows there is a natural way to produce a DFAO which generates the sequence which solves the zip specification. Lastly, using a final coalgebra one has a basis for determining equality of solutions to different zip specifications. We begin by developing this last step for automatic arrays.

3.1 Final coalgebras for automatic arrays

Denote by $\mathrm{Ar}_\Delta = \Delta^{\omega^2}$ the set of all quarter-infinite arrays with symbols in Δ. Let $c : \mathrm{Ar}_\Delta \to \Delta$ be the function taking an array to its corner entry. That is, $c(g) = g_{0,0}$. We will be showing that Ar_Δ with the transition structure given by the function c and the array projections π defined above make it into a final coalgebra.

Suppose we have a functor $FX = \Delta \times X^{kl}$ and $(G, s : G \to \Delta \times G^{kl})$ is a coalgebra for that functor. Then we name the $kl + 1$ components of the structure:

$$s = \langle o, s_{(0,0)}, s_{(0,1)}, \ldots, s_{(0,l-1)}, s_{(1,0)}, \ldots, s_{(k-1,l-1)} \rangle$$

where $o : G \to \Delta$ and $s_{(i,j)} : G \to G$. As before, o is the observation component of the structure, while the other components are the transition components of the structure. We say a coalgebra for this structure is $\mathbf{zero-consistent}$ when $o = o \circ s_{(0,0)}$, which is to say applying the zero-transition doesn't change the observation. For general coalgebras, the zero-consistency condition is a restrictive requirement, but for automatic sequences, where we know we can always modify our DFAOs to accept input with leading zeroes, zero-consistency is not a strong requirement [7].

We will also employ a notation for writing out the composition of many transitions in these coalgebras. Rather than writing $s_3 \circ s_2 \circ s_3 \circ s_1$, we will write $s_{(3)(2)(3)(1)}$, where the subscript is a word in the available transition subscripts. So, for example, we might write $(\pi_{\frac{0}{2},\frac{0}{2}} \circ \pi_{\frac{1}{2},\frac{0}{2}} \circ \pi_{\frac{0}{2},\frac{1}{2}})(x) = \pi_{(\frac{0}{2},\frac{0}{2})(\frac{1}{2},\frac{0}{2})(\frac{0}{2},\frac{1}{2})}(x)$.

Before proving that Ar_Δ is a final coalgebra, we prove two simple facts.

Lemma 3.1 *Let $(m, n)_{k,l}$ be the standard k, l-encoding of the pair (m, n). Let $i \in \mathbb{N}_{<k}$ and $j \in \mathbb{N}_{<l}$. Then $(m, n)_{k,l}(i, j)$, the encoding of (m, n) followed by the symbol (i, j), is the k, l-encoding of $(mk + i, nl + j)$*

Proof. Consider $(mk + i, nl + j)_{k,l}$. It is clear that $[mk + i]_k = [m]_k i$ and also that $[nl + j]_l = [n]_l j$. Then the last pair in the standard k, l-encoding will be (i, j), and the preceding symbols will just be the k, l-encoding of (m, n). That is, $(mk + i, nl + j)_{k,l} = (m, n)_{k,l}(i, j)$, as desired □

Lemma 3.2 *For all $a \in Ar_\Delta$, $(c \circ \pi_{(m,n)_{k,l}})(a) = a_{m,n}$.*

Proof. We show this by induction on the length of the k,l-coding for (m,n). If the encoding is empty, then $m = n = 0$ so we have $c(a) = a_{0,0}$, which is true by the definition of c.

Now suppose this statement is true for all encodings of length $\leq d$ and suppose the length of $(m,n)_{k,l}$ is $d+1$. Further let $m = q_1k + i$ and $n = q_2l + j$ where $i \in \mathbb{N}_{<k}$ and $j \in \mathbb{N}_{<l}$ by the division algorithm. There is at least one symbol in our encoding, so we write $(m,n)_{k,l} = w(i,j)$ where (i,j) is a single symbol in $\mathbb{N}_{<l} \times \mathbb{N}_{<k}$ and $w = (q_1,q_2)_{k,l}$ is the remainder of the coding by Lemma 3.1. Then we are considering $(c \circ \pi_{(m,n)_{k,l}})(a) = (c \circ \pi_w \circ \pi_{\frac{i}{k},\frac{j}{l}})(a)$ by definition of the structure subscripts. Note that w has length d, so the induction hypothesis kicks in and tells us $c \circ \pi_w$ finds the (q_1,q_2) element of its argument, so the above reduces to $(\pi_{\frac{i}{k},\frac{j}{l}}(a))_{q_1,q_2}$. Now the definition of $\pi_{\frac{i}{k},\frac{j}{l}}(a)$ above tells us the q_1,q_2 entry in this array is the $(kq_1 + i, lq_2 + j) = (m,n)$ entry of a, as desired. □

We are now ready to prove the promised result regarding the finality of the coalgebra of Ar_Δ with c and the $\pi_{\frac{i}{k},\frac{j}{l}}$.

Proposition 3.3 *The coalgebra $\langle c, \pi_{\frac{\cdot}{k},\frac{\cdot}{l}}\rangle : Ar_\Delta \to \Delta \times (Ar_\Delta)^{kl}$ is final for the zero-consistent coalgebras of the functor $FX = \Delta \times X^{kl}$.*

Proof. Suppose $\langle o, p_{(\cdot,\cdot)}\rangle : A \to \Delta \times A^{kl}$ is a zero-consistent F-coalgebra. We define a map $\varphi : A \to \mathrm{Ar}_\Delta$ by the following: $a \in A$ gets mapped to the Δ-array whose (m,n) entry is given by $\varphi(a)_{m,n} = o(p_{(m,n)_{k,l}}(a))$.

We must show that this is an F-coalgebra morphism. For this we verify the observation and transition parts separately.

$$\begin{tikzcd}
A \arrow[r, "{\langle o,p_{(\cdot,\cdot)}\rangle}"] \arrow[d, "\varphi"'] & \Delta \times A^{kl} \arrow[d, "id_\Delta \times \varphi^{kl}"] \\
\mathrm{Ar}_\Delta \arrow[r, "{\langle c,\pi_{\frac{\cdot}{k},\frac{\cdot}{l}}\rangle}"'] & \Delta \times (\mathrm{Ar}_\Delta)^{kl}
\end{tikzcd}$$

(Observation component)

We must show $o(a) = c(\varphi(a))$. We know $c(\varphi(a)) = \varphi(a)_{0,0} = o(p_{(0,0)}(a))$ by our definition of φ. Further, $o(p_{(0,0)}(a)) = o(a)$ since A is a zero-consistent F-coalgebra. Hence we have $c(\varphi(a)) = o(a)$, as desired.

(Transition components)

We must show $\varphi \circ p_{(i,j)} = \pi_{\frac{i}{k},\frac{j}{l}} \circ \varphi$ for all $i \in \mathbb{N}_{<k}$ and $j \in \mathbb{N}_{<l}$.

We first claim that $c \circ \varphi \circ p_{(m,n)_{k,l}} = c \circ \pi_{(m,n)_{k,l}} \circ \varphi$ for all $m,n \geq 0$. Note that $\varphi(a)_{m,n} = (o \circ p_{(m,n)_{k,l}})(a)$ by our definition of φ. We also know from Lemma 3.2 that $c \circ \pi_{(m,n)_{k,l}}$ finds the (m,n) element of an array, so $\varphi(a)_{m,n} = (c \circ \pi_{(m,n)_{k,l}} \circ \varphi)(a)$. Therefore $o \circ p_{(m,n)_{k,l}} = c \circ \pi_{(m,n)_{k,l}} \circ \varphi$. Now from the observation result above, we know $c \circ \varphi = o$, so we get $c \circ \varphi \circ p_{(m,n)_{k,l}} = c \circ \pi_{(m,n)_{k,l}} \circ \varphi$, and our claim is proven.

Now we return to proving $\varphi \circ p_{(i,j)} = \pi_{\frac{i}{k},\frac{j}{l}} \circ \varphi$. Let $a \in A$ and $m,n \in \mathbb{N}$. Then we must show $(\varphi \circ p_{(i,j)})(a)_{m,n} = (\pi_{\frac{i}{k},\frac{j}{l}} \circ \varphi)(a)_{m,n}$ for all such a,m,n. Since $c \circ \pi_{(m,n)_{k,l}}$

selects the (m, n) entry of a grid by Lemma 3.2, we rewrite this as

$$\begin{aligned}(c \circ \pi_{(m,n)_{k,l}} \circ \varphi \circ p_{(i,j)})(a) &= (c \circ \pi_{(m,n)_{k,l}} \circ \pi_{\frac{i}{k},\frac{j}{l}} \circ \varphi)(a) \\ &= (c \circ \pi_{(mk+i,nl+j)_{k,l}} \circ \varphi)(a)\end{aligned}$$

where the equality is by Lemma 3.1. Now using the claim we just proved we can rewrite the left hand side as

$$(c \circ \pi_{(m,n)_{k,l}} \circ \varphi \circ p_{(i,j)})(a) = (c \circ \varphi \circ p_{(m,n)_{k,l}(i,j)})(a) = (c \circ \varphi \circ p_{(mk+i,nl+j)_{k,l}})(a)$$

which is again by Lemma 3.1. This shows φ is a coalgebra map.

To show φ is the final coalgebra map, we must also show it is the *unique* map with these properties. This is a straightforward induction argument where most of the hard work is done by our previous lemmas. Since we do not need the details later, we omit them. □

3.2 Array linearization with final stream coalgebras

In the last section we showed that the set of arrays in Δ, Ar_Δ, carries a final coalgebra structure for the zero-consistent coalgebras of the functor $FX = \Delta \times X^{kl}$. We also know the stream coalgebra $(\Delta^\omega, \langle \texttt{hd}, \mathcal{N}_{kl} \rangle) = (\Delta^\omega, \langle \texttt{hd}, \pi_{0,kl}, \ldots \pi_{kl-1,kl} \rangle)$ is a final coalgebra for the zero-consistent coalgebras of F. [6] Consequently, these two coalgebras must be isomorphic. We shall next make this isomorphism explicit.

The coalgebra $(\Delta^\omega, \langle \texttt{hd}, \mathcal{N}_{kl} \rangle)$ has the structure maps $\texttt{hd}(\sigma) = \sigma_0$ and $\pi_{j,kl}(\sigma_n) = \{\sigma_{nkl+j}\}_n$ being the stream kernel maps with $0 \leq j < kl$. Now fortunately, our result above gives the explicit construction of the final coalgebra map from this coalgebra into the Ar_Δ coalgebra. Let $\varphi : \Delta^\omega \to \text{Ar}_\Delta$ be this final map. Then the array coalgebra maps, when listed out, are

$$\langle c, \pi_{\frac{0}{k},\frac{0}{l}}, \pi_{\frac{0}{k},\frac{1}{l}}, \ldots, \pi_{\frac{0}{k},\frac{l-1}{l}}, \pi_{\frac{1}{k},\frac{0}{l}}, \ldots, \pi_{\frac{k-1}{k},\frac{l-1}{l}} \rangle$$

while the stream coalgebra maps are

$$\langle \texttt{hd}, \pi_{0,kl}, \pi_{1,kl}, \ldots, \pi_{kl-1,kl} \rangle$$

so taking the corner of an array corresponds exactly to taking the head of the related stream, and the array projection $\pi_{\frac{i}{k},\frac{j}{l}}$ corresponds exactly to the stream projection $\pi_{il+j,kl}$. Now if we're given a stream σ and want to find the array $\varphi(\sigma)$, we'll try to find the element of the array at position (m, n) for all $m, n \in \mathbb{N}$. Lemma 3.2 tells us that to find the element in the array at this position we should take the $(m, n)_{k,l}$ encoding of (m, n) and follow those array projections. Since φ is an isomorphism, we know that following these projections on $\varphi(\sigma)$ corresponds exactly to taking the related stream projections on σ and then taking the head of the resulting stream.

For simplicity, let's consider the case where $k = l = 2$ and $\sigma = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, \ldots)$, and figure out what $\varphi(\sigma)$ looks like. Working through the calculation outlined above many times shows that $\varphi(\sigma)$ begins like

42	43	46	47	58	59	62	63
40	41	44	45	56	57	60	61
34	35	38	39	50	51	54	55
32	33	36	37	48	49	52	53
10	11	14	15	26	27	30	31
8	9	12	**13**	24	25	28	29
2	3	6	7	18	19	22	23
0	1	4	5	16	17	20	21

Now since Δ^ω with the hd and $\pi_{i,4}$ stream projections is also a final coalgebra for the functor $FX = \Delta \times X^4$, we know the coalgebra morphism φ must be an isomorphism. In particular, φ^{-1} takes an array and transforms it into a stream. Having figured out what φ does to a stream, we can easily invert it to see how φ^{-1} linearizes an array.

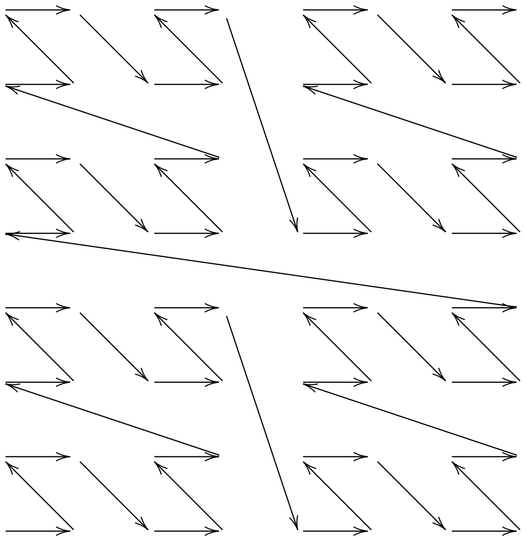


Fig. 1. The start of $\mathtt{lin}_{2,2}$

This is already known to computer scientists as the "z-order curve" as introduced by G. Morton [8]. It has an important property which makes it easy for machines working with binary representations to use: the entry at (m, n) in the array gets mapped to the element in the stream at the position which is formed by zipping the digits of $[m]_2$ and $[n]_2$. We see this on the diagram below, which is $\varphi(\sigma)$ as above but with everything translated into binary and with row and column labels. [6]

[6] We have colored all the column-related digits red, so grayscale copies of this document may appear to have a lighter shade for these digits.

111	101010	101011	101110	101111	111010	111011	111110	111111
110	101000	101001	101100	101101	111000	111001	111100	111101
101	100010	100011	100110	100111	110010	110011	110110	110111
100	100000	100001	100100	100101	110000	110001	110100	010101
011	001010	001011	001110	001111	011010	011011	011110	011111
010	001000	001001	001100	**001101**	011000	011001	011100	011101
001	000010	000011	000110	000111	010010	010011	010110	010111
000	000000	000001	000100	000101	010000	010001	010100	010101
	000	001	010	011	100	101	110	111

Indeed we might have guessed this from our definition of the automata operating on pairs of digit representations. If we look at the pairs of red and black digits, they give the pairs in the encoding of that position in the array. At the position $(2,3)_{2,2} = (1,1)(0,1) = (0,0)(1,1)(0,1)$, highlighted in bold, we indeed get the thirteenth element of the stream and $001101 = [13]_2$.

With this operation in mind, it is easy to describe what φ does as well. Given an element in a sequence, we take its position and write that position in binary, padding with leading zeroes to make the representation have an even number of digits. Then every other digit starting with the first forms the representation for the row in the array, and every other digit starting with the second forms the representation of the column in the array. For example, the 28th element of the stream gets mapped by φ to the $(2,6)$ element of the array since $[28]_2 = 11100 = 011100 = 011100$ and $010 = [2]_2$ and $110 = [6]_2$.

3.3 Connections between sequences and arrays using linearization

To avoid confusion between φ and φ^{-1}, we'll call $\varphi = \texttt{ord}$ since it takes a sequence and z-orders it into a grid and we'll call $\varphi^{-1} = \texttt{lin}$ since it linearizes an array. Each of these should be subscripted with k and l when the grid coalgebra they refer to is unclear from context.

As an example application of this isomorphism, consider the following proposition.

Proposition 3.4 *Suppose $\sigma \in \Delta^\omega$ is a sequence and $g \in Ar_\Delta$ is a grid. g is k,l-automatic iff $\texttt{lin}_{k,l}(g)$ is kl-automatic, and σ is kl-automatic iff $\texttt{ord}_{k,l}(\sigma)$ is k,l-automatic.*

Proof. We prove the first statement, regarding g and $\texttt{lin}(g)$. This will then give the second part by taking $g = \texttt{ord}(\sigma)$ and noting that $\texttt{lin}(\texttt{ord}(\sigma)) = \sigma$.

The fact that $\texttt{lin} : \mathrm{Ar}_\Delta \to \Delta^\omega$ is a coalgebra morphism immediately gives $\pi_{il+j,kl} \circ \texttt{lin} = \texttt{lin} \circ \pi_{\frac{i}{k},\frac{j}{l}}$ for all $i \in \mathbb{N}_{<k}$ and $j \in \mathbb{N}_{<l}$. Then every unique image of g under repeated application of the $\pi_{\frac{i}{k},\frac{j}{l}}$ corresponds to a unique image of $\texttt{lin}(g)$ under repeated application of the $\pi_{m,kl}$. Therefore, the k,l-kernel of g is in 1-1

correspondence with the kl-kernel of $\texttt{lin}(g)$. Since an array is k,l-automatic iff its k,l-kernel is finite, and a sequence is kl-automatic iff its kl-kernel is finite, we have g is k,l-automatic iff $\texttt{lin}(g)$ is kl-automatic. □

A further illustration of the usefulness of this isomorphism is to extend the work of [6] in automatic sequences and zip specifications to automatic arrays. First we must define the analog of the **zip** function for automatic arrays.

Definition 3.5 Let the function $\mathbf{wv}_{k,l} : \mathrm{Ar}_\Delta^{kl} \to \mathrm{Ar}_\Delta$ (read "weave") be defined by

$$\mathbf{wv}_{k,l}(g_0, g_1, \ldots, g_{kl-1}) = \texttt{ord}_{k,l}(\mathbf{zip}(\texttt{lin}_{k,l}(g_0), \texttt{lin}_{k,l}(g_1), \ldots, \texttt{lin}_{k,l}(g_{kl-1}))).$$

Take the 2x2 weave as an example. Suppose

$$A = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ a_{20} & a_{21} & a_{22} & \cdots \\ a_{10} & a_{11} & a_{12} & \cdots \\ a_{00} & a_{01} & a_{02} & \cdots \end{bmatrix},$$

and B, C and D are similar. Then

$$\mathbf{wv}_{2,2}(A, B, C, D) = \mathbf{wv}_{2,2}\begin{pmatrix} C & D \\ A & B \end{pmatrix} = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots \\ c_{10} & d_{10} & c_{11} & d_{11} & \cdots \\ a_{10} & b_{10} & a_{11} & b_{11} & \cdots \\ c_{00} & d_{00} & c_{01} & d_{01} & \cdots \\ a_{00} & b_{00} & a_{01} & b_{01} & \cdots \end{bmatrix}$$

Note this is the block-wise weave of these grids in the pattern suggested by the second argument style. **wv** is similar in spirit to **zip** since it inverts the action of the k,l-kernel maps. That is,

$$\pi_{\frac{i}{k},\frac{j}{l}}\left(\mathbf{wv}\begin{pmatrix} g_{k-1,0} & \cdots & g_{k-1,l-1} \\ \vdots & & \vdots \\ g_{0,0} & \cdots & g_{0,l-1} \end{pmatrix}\right) = g_{i,j}$$

and conversely

$$\mathbf{wv}\begin{pmatrix} \pi_{\frac{k-1}{k},\frac{0}{l}}(g) & \cdots & \pi_{\frac{k-1}{k},\frac{l-1}{l}}(g) \\ \vdots & & \vdots \\ \pi_{\frac{0}{k},\frac{0}{l}}(g) & \cdots & \pi_{\frac{0}{k},\frac{l-1}{l}}(g) \end{pmatrix} = g.$$

Now we are ready to develop a notion of a weave specification for arrays. Let S be a finite set of variables. We define a **weave term** in S by the BNF grammar

$$W ::= s \mid a : W \mid \mathbf{wv}_{k,l}(W, \ldots, W)$$

where s is a variable from S, $a \in \Delta$, $k,l \in \mathbb{N}_{\geq 2}$ and there are kl weave terms provided as the argument to $\mathbf{wv}_{k,l}$. A **weave specification** is a pairing of each variable in S to a weave term in S.

We have described how to interpret $\mathbf{wv}_{k,l}(W, \ldots, W)$. $a : W$ is to be interpreted as the grid $\mathtt{ord}(a : \mathtt{lin}(W))$. As a result of these definitions and the fact that they have isomorphic generating grammars, every weave term is the z-ordering of a zip term. This means weave specifications will have the same existence and uniqueness conditions on their solutions as their related zip specifications.

For example, the weave specification:

$$\begin{cases} m = 0 : x \\ x = 1 : \mathbf{wv}\begin{pmatrix} x & y \\ 1 : y & 0 : x \end{pmatrix} \\ y = 0 : \mathbf{wv}\begin{pmatrix} y & x \\ 0 : x & 1 : y \end{pmatrix} \end{cases}$$

has the unique solution given by the 2,2-automatic array from section 2.5. As a result of the fact that all weave terms are the z-ordering of a zip term, we can translate between the two formats easily. This gives the following theorem immediately.

Proposition 3.6 *For grids $g \in Ar_\Delta$ the following are equivalent:*

(i) g is k, l-automatic.

(ii) g can be defined by a $\mathbf{wv}_{k,l}$ specification.

(iii) g has a finite k, l-kernel

Proof. We have already noted the equivalence of (i) and (iii) is known in the literature [2]. Every $\mathbf{wv}_{k,l}$ specification is the z-ordering of a $\mathbf{zip}_{kl}$ specification, and similarly every $\mathbf{zip}_{kl}$ specification can be written as the linearization of a $\mathbf{wv}_{k,l}$ specification. Our earlier proposition shows that a grid is k, l-automatic iff its linearization is kl-automatic. Therefore, the equivalence of (i) and (ii) is the same as the equivalence of a sequence being kl-automatic and having a $\mathbf{zip}_{kl}$ specification. The latter equivalence is proven in [6], so our result follows. □

4 Zip-mix specifications

The results from the previous section suggest that a large portion of the theory of automatic sequences can be lifted directly to statements about automatic arrays by means of the z-ordering isomorphism. This, in turn, may suggest that the theory of automatic arrays may be only a reflection of the theory of automatic sequences, unremarkable in its own right. In this section we attempt to dispel this notion by using automatic arrays to solve a problem for which automatic sequences do not immediately suffice.

4.1 A subclass of zip-mix specifications

In this section we will briefly study a very restricted subclass of zip-mix specifications. The question of whether there was a general algorithm for deciding whether two sequences defined by zip-mix specifications remained open at the end of [6] and progress was recently made in [5]. We pursue this example primarily to illustrate how automatic arrays can be used in a place where automatic sequences are limited. As a step towards resolving the question of [6], it probably does not represent serious progress.

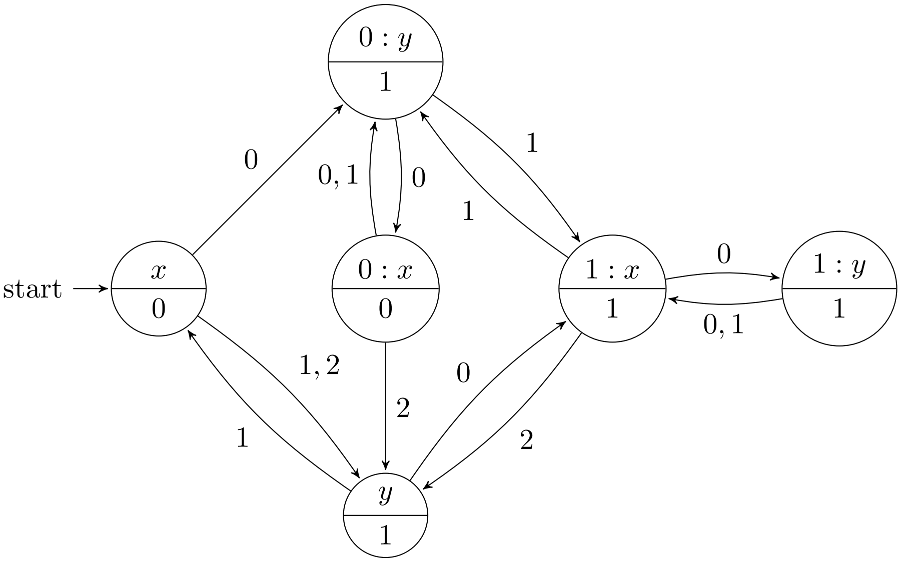
To be exact, we consider zip-mix specifications with three properties: 1) the set of variables for the specification is partitioned into two pieces $V = V_k + V_l$, 2) if $x \in V_k$ then the term for x in the specification (i.e. T_x so that $x = T_x$ is in the specification) is a zip-k term and the variables used in that term all come from V_l, and 3) similarly the term for $y \in V_l$ must be a zip-l term mentioning only variables from V_k. As an example, consider the following:

$$\begin{cases} x = 0 : \mathbf{zip}_3(y, y, y) \\ y = 1 : \mathbf{zip}_2(x, x) \end{cases}$$

In this case, $\{x\} = V_3$ and $\{y\} = V_2$, which partitions the set of variables. The term defining x is a zip-3 term which only mentions variables from V_2 and similarly the term defining y is a zip-2 term which only mentions variables from V_3, so this specification has all three properties required. We call such a zip-specification a **zip-mix specification of alternating arity**, but so as not to have to repeat this name too often, we assume all zip-specifications in this section have these properties unless specified otherwise.

If we were to apply the procedure from section 2.4 to create a DFAO for this specification, we would want to use the 3-kernel maps when looking at the state x and the 2-kernel maps when looking at the state y. That would mean we would need to have three different transitions out of the state labelled x and two different transitions out of the state labelled y. Ordinary DFAOs do not allow this, so we must alter our definition of DFAO slightly to accommodate. We define a **k,l-alternating DFAO** to be a DFAO with the following modifications: $A = \mathbb{N}_{<k} + \mathbb{N}_{<l}$, $Q = Q_k + Q_l$, $q_0 \in Q_l$ and $\delta : (Q_k \times \mathbb{N}_{<k}) + (Q_l \times \mathbb{N}_{<l}) \to Q$ must have the property that $\delta(Q_k \times \mathbb{N}_{<k}) \subseteq Q_l$ and conversely $\delta(Q_l \times \mathbb{N}_{<l}) \subseteq Q_k$. The intuition here is that we've partitioned the set of states into two disjoint pieces, Q_k and Q_l, analogous to requirement (1) on our zip-specifications. While in Q_k we read a digit from base k (i.e. from $\mathbb{N}_{<k}$) and then transition to a state in Q_l and then vice versa, analogous to requirements (2) and (3) on our zip-specifications.

Now we can create a 2,3-alternating DFAO from the zip-mix specification given above using the process described in section 2.4:



Now one way to build up the machinery needed to give an algorithm for deciding whether the solutions to two zip-mix specifications of alternating arity are equal would be to follow the general outline from [6] which we used for weave specifications earlier. First we would have to figure out how to represent this type of machine as the coalgebra of a functor, and then we would have to find a final coalgebra for that functor. This is already enough of a task, but a further demerit of this approach is that at the end we will only be able to tell whether two k, l-alternating DFAOs generate the same sequence. We would not be able to use the result, for example, to decide whether a k, l-alternating DFAO and an l, k-alternating DFAO generate the same sequence. Instead, our approach will leverage the input flexibility of k, l-DFAOs (the automata used to generate automatic arrays) along with our previous results about k, l-DFAOs.

4.2 k, l-alternating DFAOs and k, l-DFAOs

For every finite input string $\sigma = \sigma_1 \dots \sigma_{n-1}\sigma_n$ to a k, l-alternating DFAO, we can form an input to a k, l-DFAO by making the length of σ even by possibly adding a leading 0 and then pairing digits. We say a k, l-alternating DFAO and a k, l-DFAO "have the same output on input σ" if the two machines have the same output when the k, l-alternating DFAO is given σ and the k, l-DFAO is given the paired version of σ. We say these two machines of these types "have the same behavior" if they have the same output on every valid input string.

Proposition 4.1 *For every zero-consistent*[7] *k, l-alternating DFAO there is a k, l-DFAO which has the same behavior.*

Proof. Suppose ($\mathbb{N}_{<k} + \mathbb{N}_{<l}$, $Q_k + Q_l$, q_0, δ, Δ, f) is a k, l-alternating DFAO. We take the alphabet for our k, l-DFAO to be $\mathbb{N}_{<k} \times \mathbb{N}_{<l}$, the set of states to be Q_l with the same start state, the output alphabet remains Δ and the final output map is just the restriction of the original output map to our set of states, $f|_{Q_l}$. The

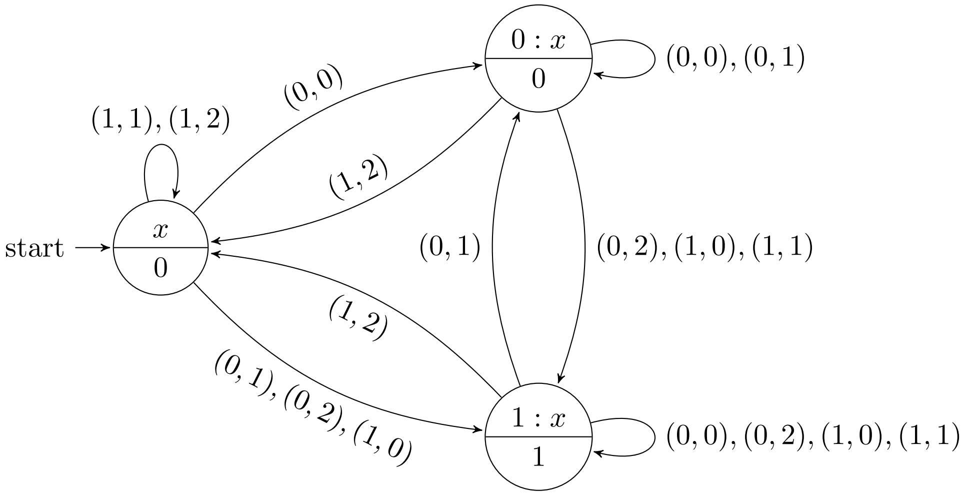
[7] As noted before, insisting on zero-consistency for DFAOs is not a strong requirement.

interesting part is the transition map for our k, l-DFAO:

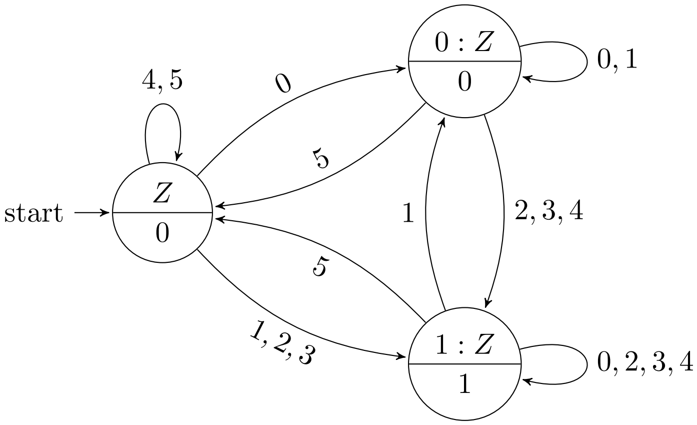
$$\delta'(q, (i, j)) = \delta(\delta(q, j), i)$$

With this definition it is easy to check that the extended transition functions of these DFAOs coincide for all even-length inputs to the k, l-alternating DFAO.[8] For odd input lengths we must use the zero-consistency property of the alternating DFAO since the input to the k, l-DFAO was padded with an extra leading 0. □

If we apply this process to the 2,3-alternating DFAO given above, we get the following 2,3-DFAO



Now we can use our earlier results for automatic arrays. Since the array generated by the 2,3-DFAO is by definition an automatic array, its linearization is 6-automatic by Proposition 3.4. In fact, we can get the 6-DFAO by just taking the transitions labels above and finding the corresponding transition in the 6-kernel maps. This yields the following 6-DFAO:



This DFAO generates the sequence which is the solution for x in the zip specification above. Now imagine we had another zip specification of alternating arity. We could create a DFAO from that specification using the same steps and then to

[8] Here it is particularly important for this proof that our automata read in the backwards order, but the proof can be adapted to other input conventions.

check whether they generated the same stream we would just have to check for a bisimulation.

Proposition 4.2 *There is an algorithm which decides whether two zip-mix specifications of alternating arity have the same solution, provided the product of the two arities is the same.*

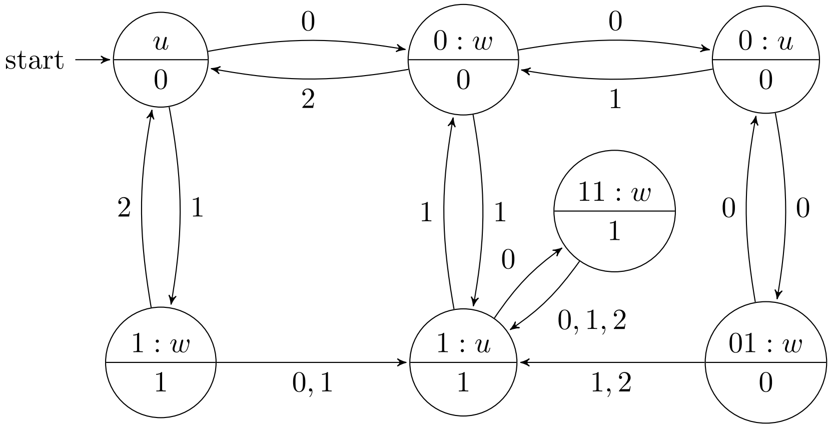
Proof. For each zip-mix specification of alternating arity, generate a k, l-alternating DFAO using the procedure in section 2.4. From the k, l-alternating DFAO we can create a k, l-DFAO with the same behavior by the recipe in Proposition 4.1. Note the solution to the zip-mix specification is the linearization of the array produced by the k, l-DFAO. From these k, l-DFAOs we can create the kl-DFAO which generates the linearization of the array by Proposition 3.4, so it suffices to check whether these two kl-DFAOs generate the same sequence. Since the product of the two arities of the zip-mix specifications is the same, we have two kl-DFAOs. These generate the same sequence iff they can be put in bisimulation with one another, for which there is also an algorithm. □

Remark 4.3 Please note that one could have zip-mix specifications of alternating arity where the arities are completely different numbers. This proposition only requires the product of the arities to be the same. For example, given a zip-mix specification where the arities are 2 and 6 and another zip-mix specification where the arities are 3 and 4, the above steps produce two 12-DFAOs which generate the solution to each specification separately. These solutions are equal iff the 12-DFAOs can be put in a bisimulation including the start states.

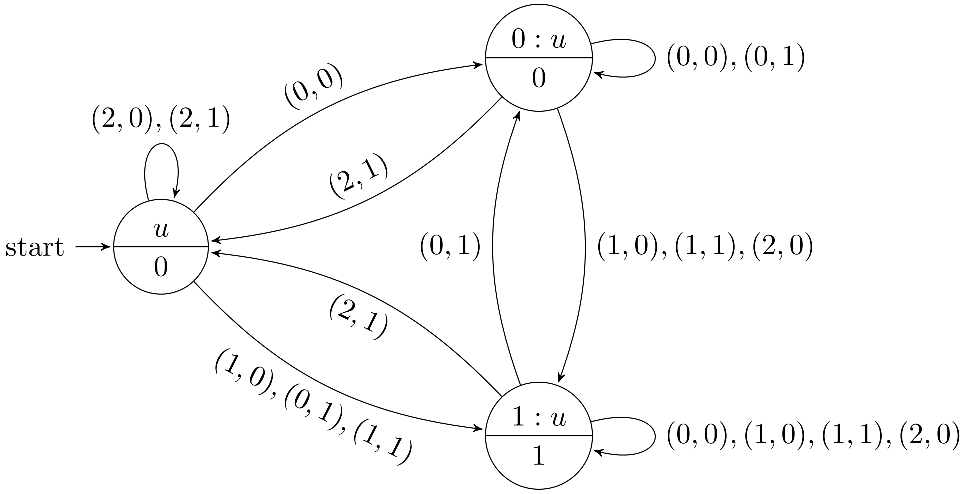
As an example of this procedure, consider the following zip-mix specification:

$$\begin{cases} u = 01 : \mathbf{zip}_2(w, w) \\ w = 1 : \mathbf{zip}_3(u, u, u) \end{cases}$$

We claim this specification has the same solution (for u) as the specification listed earlier in this section (for x). We first find the 3,2-alternating DFAO for this specification:



Then we can construct the related 3,2-DFAO as described in the above proposition.



When we create the related 6-DFAO from this 3,2-DFAO, we actually get the same 6-DFAO as we got for the other specification! Since we certainly have a bisimulation between these two machines, we can conclude that the sequence generated by them is the same and therefore the solutions to the specifications are the same.

5 Summary

We have presented a natural final coalgebra for arrays, for which automatic arrays are the rational part. This coalgebra is final for a functor also commonly used in the study of automatic sequences and hence gives rise to an isomorphism between the array coalgebra and sequence coalgebra preserving many important automaticity properties. We then lifted several results about automatic sequences to facts about automatic arrays, including the zip specification scheme of [6] to the case of automatic arrays. Finally, using the additional flexibility of automatic arrays we gave a partial answer to an open problem regarding zip specifications of mixed arity. There are a few natural avenues for investigation to proceed from here, which we summarize briefly.

m-partitions and other variadic zip specifications We assumed in section 4.1 that we had a bipartition of the terms in the specification. If we have a tripartition or, more generally, a partition of the variables into m sets with the property that everything in the same class has the same zip-arity and all variables mentioned in the terms in that set come from another class of the partition, a similar theory using an automatic complex of dimension m should yield decidability. We would like some exploration on what automata naturally emerge from relaxing this restriction on the variables and perhaps answering the decidability question for zip specifications where the base determination is made by a DFA. (See [6,5] for more details on this set up.)

Other space filling curves We found the z-order/Morton curve as an isomorphism between array coalgebras and sequence coalgebras. There are many other space filling curves (such as the Hilbert curve or the Peano curve), so we wonder whether there are coalgebras which naturally give rise to these curves as isomorphism between arrays and sequences.

Acknowledgement

I owe many thanks to Larry Moss for his invaluable advice, comments, and patience reading much rougher drafts.

References

[1] J.-P. Allouche and J. Shallit. The ubiquitous Prouhet-Thue-Morse sequence. In T. Helleseth C. Ding and H. Niederreiter, editors, *Sequences and their applications, Proceedings of SETA'98*, pages 1–16. Springer Verlag, 1999.

[2] J.-P. Allouche and J. Shallit. *Automatic Sequences.* Cambridge University Press, 2003.

[3] G. Christol. Ensembles presque periodiques k-reconnaissables. *Theoret. Comput. Sci.*, 9(1):141–145, 1979.

[4] Alan Cobham. Uniform tag sequences. *Theory of Computing Systems*, 6(1):164–192, 1972.

[5] J. Endrullis, C. Grabmayer, and D. Hendriks. Mix-automatic sequences. In *Language and Automata Theory and Applications (LATA 2013)*, pages 262–274. Springer, 2013.

[6] C. Grabmayer, J. Endrullis, D. Hendriks, J.W. Klop, and L.S. Moss. Automatic sequences and zip-specifications. In *Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer Science*, pages 335–344. IEEE Computer Society, 2012.

[7] C. Kupke and J.J.M.M. Rutten. On the final coalgebra of automatic sequences. In *Logic and Program Semantics*, pages 149–164. Springer, 2012.

[8] G.M. Morton. *A computer oriented geodetic data base and a new technique in file sequencing.* International Business Machines Company, 1966.

[9] J. J. M. M. Rutten. Automata and coinduction (an exercise in coalgebra). In *CONCUR'98: concurrency theory (Nice)*, volume 1466 of *Lecture Notes in Comput. Sci.*, pages 194–218. Springer, Berlin, 1998.

[10] J. J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theoret. Comput. Sci.*, 249(1):3–80, 2000. Modern algebra and its applications (Nashville, TN, 1996).

[11] J. J. M. M. Rutten and D. Turi. On the foundations of final semantics: nonstandard sets, metric spaces, partial orders. In *Semantics: foundations and applications (Beekbergen, 1992)*, volume 666 of *Lecture Notes in Comput. Sci.*, pages 477–530. Springer, Berlin, 1993.

Bicategorical Semantics for Nondeterministic Computation

Mike Stay[1]

Department of Computer Science, University of Auckland, New Zealand
Biosimilarity LLC, Seattle, USA

Jamie Vicary[2]

Centre for Quantum Technologies, National University of Singapore, Singapore
Department of Computer Science, University of Oxford, UK

Abstract

We present a topological bicategorical syntax for the interaction between public and private information in classical information theory. This allows high-level graphical definitions of encrypted communication and secret sharing, including a characterization of their security properties. This analysis shows that these protocols have an identical abstract form to the quantum teleportation and dense coding procedures, giving a concrete mathematical analogy between quantum and classical computing. Specific implementations of these protocols as nondeterministic classical procedures are recovered by applying our formalism in a symmetric monoidal bicategory of matrices of relations.

Keywords: Categories, security, geometry

1 Introduction

1.1 Overview

This paper describes a bicategorical language for reasoning about cryptographic processes in classical computation. Bicategories can be thought of as generalizations of monoidal categories, mathematical structures which have already found significant application to quantum computation [1,12]. In this work, we describe a monoidal bicategory **2Rel**, and describe how its different layers of structure can be used to describe public information, private information and nondeterministic classical processes in a natural way.

Bicategories have a well-known sound and complete graphical calculus [6,11] involving points, lines and regions, which we make use of almost exclusively for presenting our formalism and proving our results. The reason is that the basic axioms we impose have a

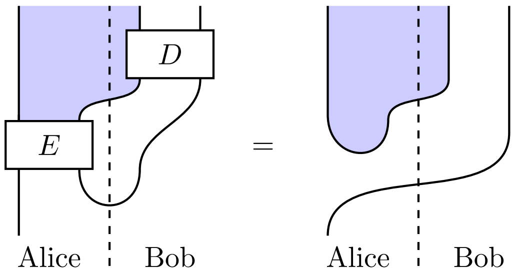
[1] Email: msta039@aucklanduni.ac.nz

[2] Email: jamie.vicary@cs.ox.ac.uk

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.com/locate/entcs

direct topological interpretation which are cumbersome from an algebraic perspective, but which the graphical calculus naturally absorbs and makes trivial. Why this should be is far from clear; it provides evidence of a deep relationship between topological structures and the theory of information [2] which deserves to be explored further.

A diagram in our calculus can he interpreted as a history of computational events, in which time flows from bottom to top. To use the terminology of physics, they are 'spacetime diagrams' for our computation. As an example of our notation, the following diagram represents an encrypted communication protocol making use of a one-time pad:



(1)

A full description of the components of these diagrams must wait for Section 3, but we can summarize the basic features here. The shaded regions represent public information, and the vertices represent computational processes. Lines represent computational systems which carry information: if the line borders a shaded region then the associated system carries a copy of the associated public information, but otherwise it carries private information. The dashed vertical lines, which are not a part of the mathematical formalism, imply a separation of the components involved between Alice and Bob which is convenient for our interpretation.

In the left-hand diagram, the bottom-left line indicates a system held by Alice which stores some private information. This information is the plaintext that we will encrypt. The bowl-shaped curve represents the first nontrivial process—the nondeterministic creation of a one time-pad—which is shared between the two parties. The next vertex E represents encryption, a process which takes as input the plaintext and a copy of the one-time pad, and produces public information. A copy of that public information is then transferred to Bob, and is fed into the decryption process D along with a copy of the one-time pad. The line emerging at the top-right represents the decrypted plaintext. The result of this entire composite procedure is given by the right-hand diagram: the original plaintext is simply transmitted unaltered, and the public information is disconnected, meaning it is uncorrelated with the plaintext. The equality between these two sides says that the encryption-decryption process is successful.

In this description, and throughout this paper, we freely interpret the underlying mathematical structures in a way which is intended to make our formalism easier to understand at an intuitive level. However, this interpretation it is secondary to the basic mathematical content of our theory, which is crisp and unambiguous. The motivating result is Theorem 4.1, which states that solutions to equation (1) in **2Rel** such that E is kernel-free correspond exactly to implementations of classical encrypted communication via a one-time pad. A variety of security properties of this procedure are also provable using our techniques.

The form of equation (1) corresponds exactly to the equation for *quantum teleportation*, as described in the bicategorical approach to quantum information [12]. One of the most important procedures in quantum theory, and yet uncovered only relatively recently [3], quantum teleportation is a procedure whereby two parties who share pre-existing quantum entanglement can transmit quantum information between them, by communicating only classical information. A strong analogy to classical encrypted communication can be drawn: two parties who share a pre-agreed secret key can transmit secret information between them, by communicating only public information. This analogy has already been recognized by several authors [5,8], and our work provides a new formal mathematical basis for it.

The monoidal bicategory **2Rel** which forms the basis for our constructions is described in Section 2. In Section 3 we gives the details of our graphical formalism, and Section 4 contains an application of our techniques to encrypted communication and secret sharing procedures, including an analysis of their security properties.

2 The Bicategory of Matrices of Relations

2.1 Introduction

Bicategories, also known as weak 2-categories, are algebraic structures akin to higher-dimensional directed graphs, and play an important role in modern mathematics. They are built from vertices, edges going between vertices, and surfaces going between edges, which are called 0-cells, 1-cells and 2-cells respectively. They also carry algebraic structure, allowing edges to be composed along a common vertex, and surfaces to be composed along a common edge. These composition operations are required to be unital and associative in a suitable fashion. Though elegant, the full definition is lengthy and we omit it here; see [4] for a good introduction.

In this section we describe the bicategory **2Rel** which will be the target for our constructions. It can be presented quite simply in terms of finite sets and partitions: 0-cells are finite sets, 1-cells are finite sets partitioned by their source and target sets, and 2-cells are relations getting along with the partitioning. All the structure of a bicategory can be defined quite naturally on these structures. We give a careful definition below, although for must purposes an intuitive understanding of the structure is quite adequate.

2.2 Construction

The n-cells of **2Rel** are defined in the following way. **0-cells** are finite sets, denoted $S, T, \ldots$. A **1-cell** $A : S \to T$ is a family of finite sets $A_{t,s}$ indexed by $s \in S$ and $t \in T$. For 1-cells $A, B : S \to T$, a **2-cell** $\rho : A \Rightarrow B$ is a family of relations $\rho_{t,s} : A_{t,s} \to B_{t,s}$ indexed by $s \in S$ and $t \in T$.

To demonstrate that these form a bicategory we first observe that for each pair of 0-cells S and T, the 1-cells $S \to T$ and the 2-cells between them form a category in a straightforward way, under ordinary relational composition of 2-cells. Identity 1-cells $\mathrm{id}_S : S \to S$ are chosen as the family $\delta_{s,s'}$, which is defined as the 1-element set if $s = s'$ and the 0-element set otherwise. Horizontal composition is a family of functors

$$\circ : \mathrm{Hom}(S,T) \times \mathrm{Hom}(T,U) \to \mathrm{Hom}(S,U) \tag{2}$$

for each ordered triple S, T, U of 0-cells. On 1-cells $A : S \to T$ and $B : T \to U$, we define this as

$$(B \circ A)_{u,s} = \coprod_{t \in T} B_{u,t} \times A_{t,s}. \tag{3}$$

This extends to 2-cells in a natural way.

The final pieces of structure are the structural 2-cells of the bicategory. For each family of composable 1-cells $A : S \to T$, $B : T \to U$ and $C : U \to V$ we require an invertible 2-cell

$$\phi_{A,B,C} : (C \circ B) \circ A \Rightarrow C \circ (B \circ A). \tag{4}$$

Writing out the source and target using definition (3), we see that ϕ is built from a family of isomorphisms $\coprod_t \left((\coprod_u C_{v,u} \times B_{u,t}) \times A_{t,s} \right) \simeq \coprod_u \left(C_{v,u} \times (\coprod_t B_{u,t} \times A_{t,s}) \right)$, each of which can be constructed canonically. Unit 2-cells can be straightforwardly defined, and it is then straightforward to show that the required pentagon and triangle equations commute.

In fact, **2Rel** can be given the structure of a symmetric monoidal bicategory, for which the tensor product of two 0-cells is their cartesian product as sets. For full details see [10], in which an equivalent bicategory **Mat**(**Rel**) is described. According to this structure, the monoidal unit 0-cell is the 1-element set.

Endomorphisms in **2Rel** have the following property, which will useful later.

Lemma 2.1 *In* **2Rel***, if 2-cells* σ *and* τ *are endomorphisms, then* $\sigma \circ \tau = \mathrm{id}$ *implies* $\tau \circ \sigma = \mathrm{id}$.

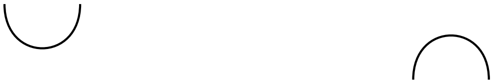
Proof. Suppose at first that σ and τ are relations on a finite set S. Then if $\sigma \circ \tau = \mathrm{id}_S$, there must be at least one $y \in S$ such that $(x, y) \in \sigma$ and $(y, x) \in \tau$. But then there must be exactly one such y, otherwise we could not ensure that $x \neq z \in S$ implies $\nexists y \in S$ with $(x, y) \in \sigma$ and $(y, x) \in \tau$. It follows that σ and τ are graphs of mutually inverse bijections, and so in particular $\tau \circ \sigma = \mathrm{id}_S$ also. The hypothesis follows immediately. □

3 Private and Public Information

3.1 Private information

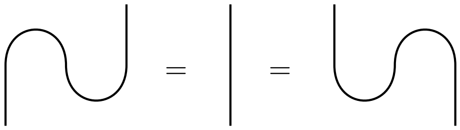
We assume that a single, isolated computational system is located at any moment at a single point in space, so that over time its history traces out a line in spacetime. In the absence of shaded regions, our diagrams are simple representations of such a scenario, with vertices representing points at which multiple systems interact. This part of our graphical formalism is the standard notation for morphisms in symmetric monoidal category [9]. Our string diagrams are valued in **Rel**, the symmetric monoidal category of finite sets and relations. This forms the endomorphisms of the 1-element set considered as a 0-cell of **2Rel**. In this way the ordinary string diagram calculus for **Rel** embeds into our surface diagram calculus for **2Rel** in a natural fashion. We will interpret an object of **Rel** as representing a classical computational system, with a particular finite set of internal states. Morphisms are arbitrary nondeterministic computational dynamics, transforming states of the domain into states of the codomain.

Using this formalism, we call a system *self-dualizable* if it can be equipped with a unit morphism and a counit morphism



(5)

satisfying the following equations, called the snake equations:

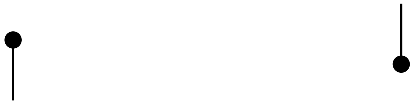


(6)

The unit morphism represents a way to create two systems together, and the counit morphism represents a way to eliminate two systems together. In essence, the snake equations say that we can choose these structures in a way that represents the topology of a string.

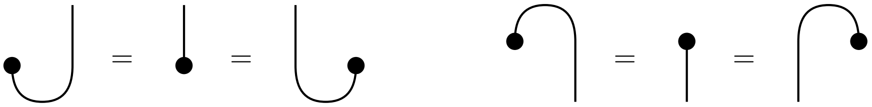
We say that the unit and counit morphisms *witness* the self-duality. In **Rel** every object A is self-dualizable, with the unit morphism $\eta : 1 \to A \times A$ given canonically by $\eta = \bigcup_{a \in A} \big(\bullet, (a,a)\big)$, and with the counit given by the converse of this relation. Every unit and counit map is of this form, up to isomorphism. The unit morphism η represents a nondeterministic processes whereby a pair of systems are prepared, each in the same state $a \in A$, such that any pair (a, a) can arise in this way. We can interpret this computationally as a *one-time pad distribution procedure.* It is deeply interesting that this should arise solely from the requirements of the snake equations (6).

For a set A there is a unique relation of type $A \to 1$ which is *total*, meaning that every element of $A \times 1$ is in the relation. It can be characterized abstractly as the unique morphism of this type with zero kernel [7], and is interpreted as eliminating the system A without halting the computation. It has a converse relation, which represents the process of creating the system A in an arbitrary state. We denote these morphisms graphically in the following way:



(7)

These are related by the unit and counit morphisms (5) witnessing self-dualizability via the following equations:

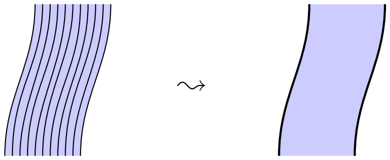


(8)

Each of these has a natural interpretation in **Rel** terms of nondeterministic classical computation: the first set of equalities (8) say that if you nondeterministically create shared keys and then delete one of the keys, the remaining key is uniformly random; while the second set say that if you have a given key, it is always possible that another key produced nondeterministically might match it.

3.2 Public information

Public information is represented in our formalism by regions. The intuitive idea is that public information is stored by a family of systems, each strongly correlated with their neighbours. Each individual system sweeps out a line through time, so the family sweeps out an entire region:

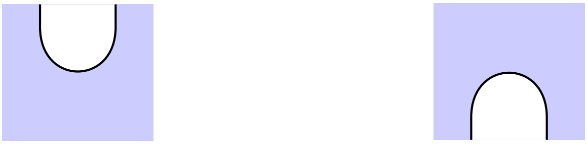


(9)

We shade this region in blue to indicate its special interpretation. The interpretation as public information derives entirely from the fact that the information is now available at many locations, each of which store an identical copy. So public information is more accessible than private information, but as a consequence less mutable, since to change its value every representative would have to be modified. This mathematical model gives a reasonable abstraction for real-world public data services, such as the Domain Name Service, which stores public information redundantly on many independent computers.

As mentioned in the introduction, we are making use here of the standard graphical calculus for monoidal bicategories. Shaded regions correspond to 0-cells of the bicategory, and unshaded regions correspond to the monoidal unit 0-cell.

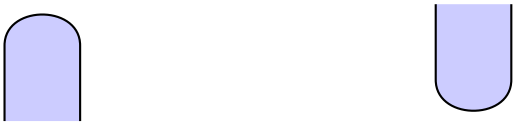
Manipulations of our public data are described by a small number of basic components. Copying and comparing public data are represented as follows:



(10)

In the first of these one region splits into two, each carrying a copy of the original public information. In the second two regions fuse to become one, which carries the same information as the initial regions in the event that the data in both initial regions is the same. Otherwise, the computation halts; in this sense, this second vertex can be interpreted as the *assertion* that two data values compare successfully.

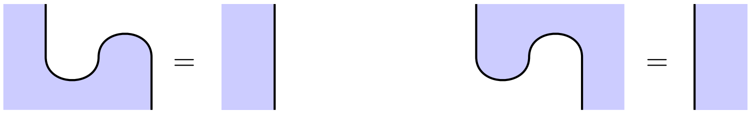
We can also represent deletion and uniform creation of public information:



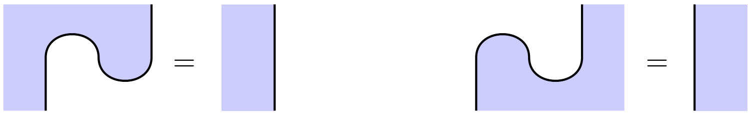
(11)

In the first of these, a single region is eliminated, deleting the information it stores. In the second, a single region is created, which we interpret as holding any possible value of the public information in a nondeterministic sense.

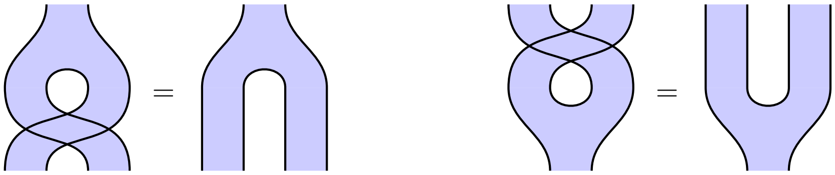
As with the bicategorical syntax for quantum information [12], in order to support their interpretations, we require these copying, deleting, comparison and uniform creation components to satisfy certain equations. They are topological, in that they amount to saying that any composite diagram built from them is determined only by its connectivity.



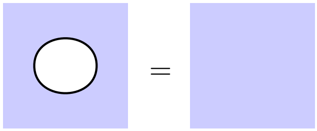
(12)



(13)



(14)



(15)

Each of these equations is consistent with the interpretation we give to the basic components (10)–(11). For example, the first equality labelled (12) asserts that copying public information and then deleting the new copy results in the identity; the first equality labelled (14) represents the fact that exchanging public information and then comparing gives the same result as simply comparing; and equation (15) states that copying public information and then immediately comparing yields the identity. In terms of higher category theory, equations (12)–(13) state that the region boundaries are ambidextrous adjoints [6], and equations (14)–(15) state that the associated Frobenius algebra is special and commutative.

The following theorem demonstrates that these structures are easy to work with in **2Rel**.

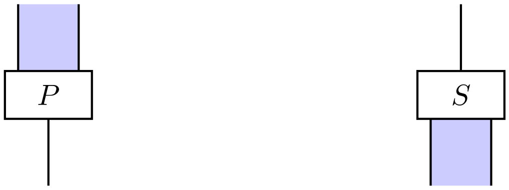
Theorem 3.1 *Every 0-cell in* **2Rel** *carries structures* (10)–(11) *satisfying equations* (12)–(15) *in an essentially unique way.*

Proof sketch. A 1-cell $A : 1 \to S$ is determined by an S-indexed family of finite sets A_s, and its isomorphism class is determined by the cardinalities of those sets. Every such 1-cell has an ambidextrous adjoint, meaning precisely that essentially unique values can be given for structures (10)–(11) that satisfy equations (12)–(13). The result is a Frobenius algebra structure [6], which will be commutative exactly when each of the finite sets A_s has cardinality 1, which satisfies the equations labelled (14). The resulting structures automatically satisfy equation (15). □

3.3 Interacting private and public data

Interesting phenomena arise when we study interactions between public and private information. There are three basic forms that such an interaction can take: converting private data to public data; converting public data to private data; and using public data to modify private data.

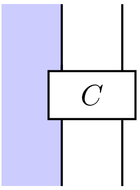
Conversion processes between public and private data take the following forms:



(16)

Here P is a publication process converting private data into public data, and S is a sampling process converting public data into private data. Their interpretation rests entirely on their types; there are no equations which we require them to satisfy. These processes need not be deterministic, or invertible, in general. However, it will later be convenient to require them to be kernel-free, meaning that they do not halt on any input.

The final type of process we introduce is the controlled computation, which performs an operation on private data in a way which depends on the value of some public data:

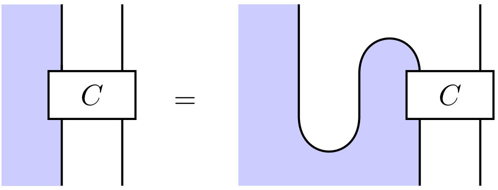


(17)

Such an operation can modify the private data, but not the public data.

Lemma 3.2 *A controlled computation cannot modify public data.*

Proof. We can use the topological behaviour of public information to rewrite our controlled computation vertex C in the following way:



(18)

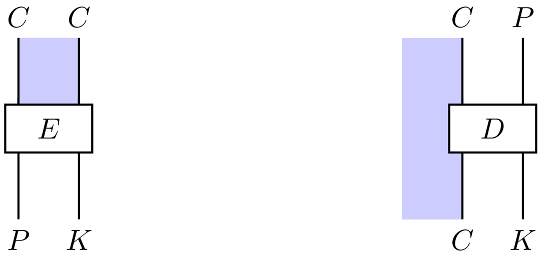
In this form it is clear that the public data is not modified, since it is explicitly copied before C is implemented. □

This result fits well with our intuition about public data as a being carried by a large, correlated family of systems. To change the value of the public data would require modifying all of these systems, but the process C only has access to a restricted subset, as made explicit by the open boundary on the left-hand side of the diagram. If the proof given here seems too slick to have any real content, that is because this is really a lemma about our *interpretation* of these mathematical structures, rather than about those structures themselves.

4 Modelling Cryptographic Procedures

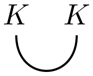
4.1 Encrypted communication

Suppose Alice is sending an encrypted message to Bob. We use a 2-cell E to represent Alice's encryption process, which relates the private plaintext P and the private key K to the public ciphertext C. Bob's decryption process is a 2-cell D that relates the public ciphertext and private key to the same ciphertext and a private plaintext. We represent these 2-cells graphically in the following way:



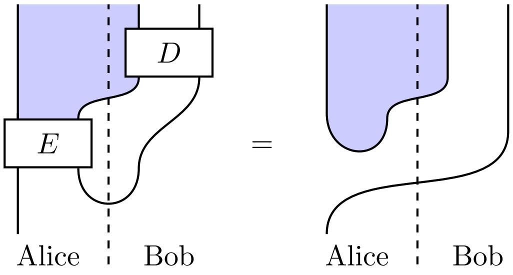
(19)

While encryption and decryption are deterministic, key generation is not. We represent key generation as a special 2-cell, the curried identity relation on the set of keys K:



This is the unit morphism for a self-duality on K, as described in Section 3.

Using our topological language, we can express correctness of encrypted communication in the following way:



(20)

This is the same 2-dimensional equation as that used in [12] to describe quantum teleportation. The encryption step takes the place of the measurement operation, and the decryption step takes the place of the controlled unitary correction. The ciphertext takes the place of the classical bits transmitted from Alice to Bob. This provides an intuition for why no faster-than-light communication is possible with entangled particles: Alice and Bob merely share a quantum variant of a one-time pad, and the actual encoded message must still be sent at some finite speed.

The following theorem is the motivation for our entire theory.

Theorem 4.1 *Solutions to* (20) *in* **2Rel** *for which E is kernel-free correspond exactly to implementations of classical possibilistic encrypted communication by a one-time pad.*

Proof sketch. The result is established by construction. From a description of a space of messages, a family of one-time pads, and encryption and decryption procedures, a solution to (20) can be directly constructed, and the inverse procedure is also possible. □

We illustrate our proof sketch with the simplest nontrivial implementation of the protocol: the encrypted communication of a single bit. We can describe concretely the values of E, D and the key creation step η as 2-cells in **2Rel** which correspond to this scenario. We choose $C = P = K$ to be the 2-element set, and we define the 2-cells as follows:

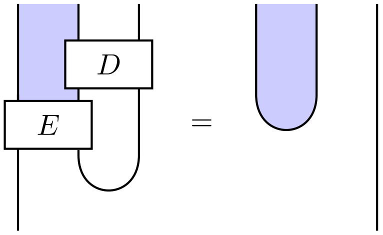
$$E = \left(\begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \right) \qquad D = \left(\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \right) \qquad \eta = \left(\begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} \right) \tag{21}$$

Here E is a matrix containing a single relation from a 4-element set to a 2-element set, which is exactly the multiplication operation for the group $\mathbb{Z}_2$; D is matrix of invertible single-bit operations to apply depending on which bit is published at the encryption step; and η is a matrix with a single entry, the relation representing nondeterministic creation of the pair of keys $(0, 0)$ or $(1, 1)$. Using the definition of the bicategory **2Rel**, it can be checked that these values satisfy equation (20).

4.2 *Security properties*

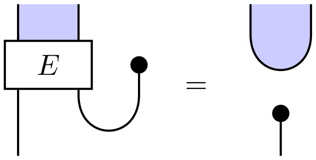
Our formalism allows us to prove results about the protocol based on only its abstract form, and hence draw conclusions which will apply for any implementation. Many of these results can be naturally interpreted as describing security properties. The generality of our results means that we can presume an attacker with arbitrary computational abilities, as long as their actions are constrained to those that can be described using our formalism (i.e. arbitrary nondeterministic processes.)

To focus on its algebraic properties, we simplify equation (20) topologically in the following way:



(22)

The first property we will examine can be considered the primary security property for encrypted communication:



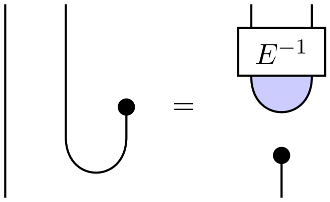
(23)

This says that if we encrypt a message using one copy of a one-time pad, and then delete the other copy of the one-time pad, this is equivalent to deleting our original message and producing a random ciphertext. So the ciphertext carries no information about the plaintext in the absence of the private key.

We can use our formalism to derive from this security property a strong constraint on the encryption operation E.

Theorem 4.2 *If the encryption step in classical encrypted communication satisfies property (23), then encryption is not invertible unless the space of messages is trivial.*

Proof. Suppose encryption is invertible. Then composing both sides of (23) with E^{-1} gives the following graphical expression:



(24)

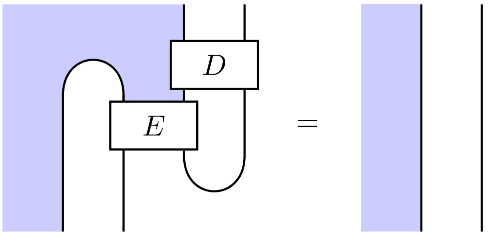
Hence the identity process on the set of messages factors through the one-element set. □

This is a desirable property: if encryption were invertible, then both the plaintext and the secret key would be derivable in principle from the ciphertext.

We can draw a very different conclusion for the decryption process D.

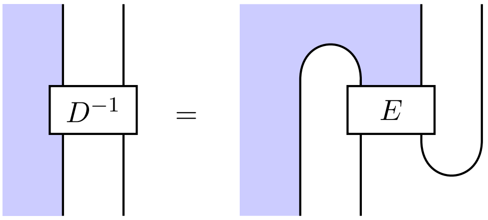
Theorem 4.3 *In classical encrypted communication, the decryption step is invertible.*

Proof. From equation (22) representing correctness of encrypted communication, we apply the topological properties of public information to obtain the following equivalent equation:



(25)

This says that D has a right inverse given by E with its top-left and bottom-right legs twisted in the manner indicated. However, by Theorem 2.1, if an endomorphism is a left inverse then it must also be a right inverse, and hence our theorem follows, with the following expression for D^{-1}:

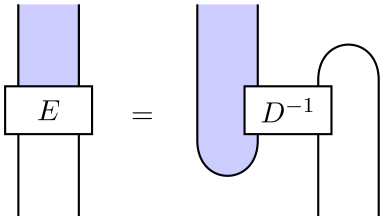


(26)

□

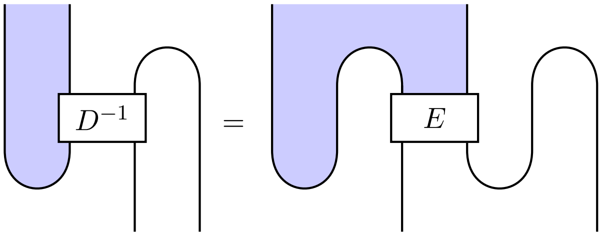
It follows that we can reconstruct E from the knowledge of D and its inverse.

Theorem 4.4 *For an implementation of classical encrypted communication, we have*



(27)

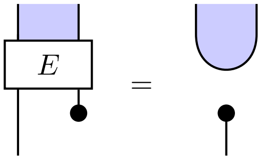
Proof. We apply the topological properties of public information to expression (26) to obtain the following:



(28)

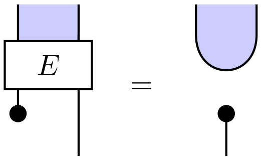
The right-hand side of this expression evaluates to E, by the topological properties (12) of 2-dimensional regions and the snake equations (6). □

While property (23) is primary, there are other security properties of the encryption process that we could consider. The first states that if we encode with a random key, this is equivalent to deleting the original message and producing random ciphertext:



(29)

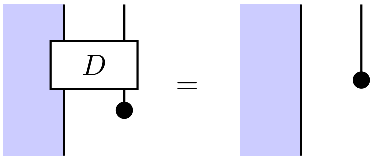
Secondly, we could encode a random message with a specified key:



(30)

This property says that this is the same as deleting the key, and producing a random ciphertext.

We can also consider security properties for the decryption process.



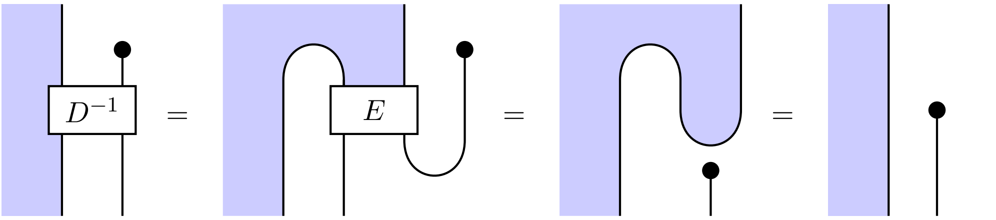
(31)

This says that if an attacker chooses nondeterministically from the space of all possible keys, every possible message can be produced, regardless of the ciphertext. So if an attacker has no knowledge of the key, they cannot extract information from the ciphertext.

In fact, we can use our formalism to show that all of these security properties follow from the primary security property (23).

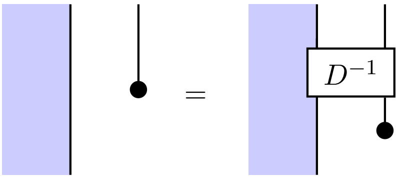
Theorem 4.5 *In classical encrypted communication,* (23) *implies* (29)*,* (30) *and* (31)*.*

Proof. The implication (23) ⇒ (29) follows from the topological property (8) of the deletion map. For the other implications, we compose expression (26) for D^{-1} with the deletion map at the top-right leg, obtaining the following:



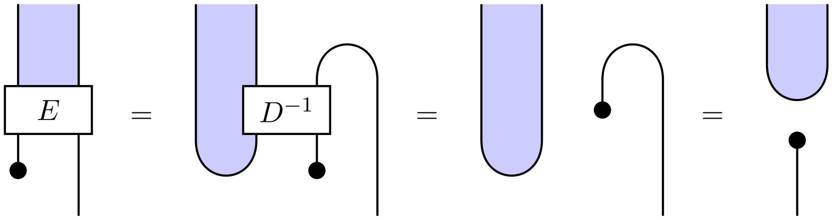
(32)

Every invertible 2-cell in **Rel** is a family of bijections, and hence its converse is its inverse. Taking the converse is a functorial operation, and so taking the converse of of the first and last diagram here, we obtain property (31). For the final property (30), we postcompose this expression with the 2-cell D^{-1}, obtaining the following expression:



(33)

We can use this to prove security property (30), where we also make use of expression (27) giving E in terms of D^{-1}:

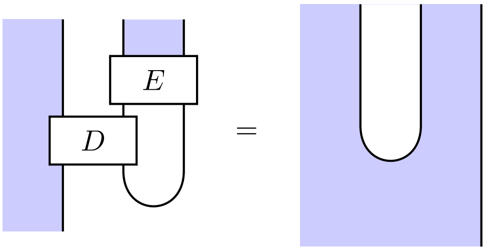


(34)

This completes the proof. □

4.3 Secret sharing

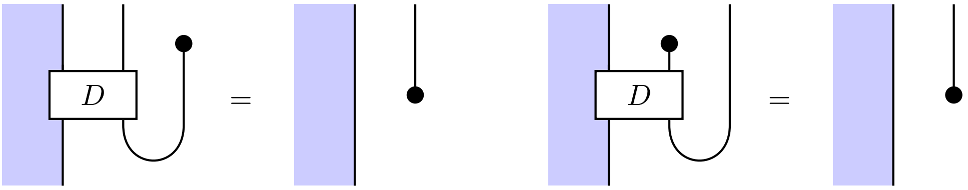
We can represent correctness of a secret sharing procedure in the following way:



(35)

On the left, we begin with some pre-existing public information. This is the information to be communicated by the procedure. We prepare two correlated systems using a one-time pad, and then manipulate the first copy by a procedure D that depends on the value of the classical data. The result is a pair of ciphertexts. Both are then consumed together by a process E, producing public information. This is successful when the result is to copy the original public information.

The important security property of a secret sharing procedure is that if only one ciphertext is available, then no information about the original message can be regained. A strong, constructive way to phrase this is to say that if one of the ciphertexts is erased, the other becomes uniformly random, and independent of the original message. This gives two conditions, with the following graphical representations:



(36)

Equation (35) has an identical structure to the quantum dense coding equation given in [12], and the security properties (36) are equivalent to properties (31) and (33) of the encrypted communication protocol.

References

[1] Samson Abramsky and Bob Coecke. A categorical semantics of quantum protocols. *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science*, pages 415–425, 2004. IEEE Computer Science Press.

[2] John Baez and Michael Stay. *Physics, Topology, Logic, and Computation: A Rosetta Stone*, volume 813, pages 95–172. Springer, 2011.

[3] Charles H. Bennett, Gilles Brassard, Claude Crépeau, Richard Jozsa, Asher Peres, and William K. Wootters. Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. *Physical Review Letters*, 70(13):1895–1899, 1993.

[4] Francis Borceux. *Handbook of Categorical Algebra: Volume 1*. CUP, 1994.

[5] Daniel Collins and Sandu Popescu. A classical analogue of entanglement. *Physical Review A*, 65(3), 2001.

[6] Aaron D. Lauda. Frobenius algebras and ambidextrous adjunctions. *Theory and Applications of Categories*, 16(4):84–122, 2006.

[7] Saunders Mac Lane. *Categories for the Working Mathematician*. Springer, 1997. 2nd edition.

[8] Jonathan Oppenheim, Rob Spekkens, and Andreas Winter. A classical analogue of negative information. 2005.

[9] Peter Selinger. *New Structures for Physics*, chapter A Survey of Graphical Languages for Monoidal Categories, pages 289–355. Springer, 2011.

[10] Mike Stay. Compact closed bicategories. 2013. arXiv:1301.1053.

[11] Ross Street. Low-dimensional topology and higher-order categories. In *Proceedings of Category Theory 2005*, 1995.

[12] Jamie Vicary. Higher semantics of quantum protocols. *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science*, 2012.

A Stream Calculus of Bottomed Sequences for Real Number Computation

Kei Terayama[1] Hideki Tsuiki[2]

Graduate School of Human and Environmental Studies
Kyoto University
Kyoto, Japan

Abstract

A calculus XPCF of $1\bot$-sequences, which are infinite sequences of $\{0, 1, \bot\}$ with at most one copy of bottom, is proposed and investigated. It has applications in real number computation in that the unit interval $\mathbb{I}$ is topologically embedded in the set $\Sigma^{\omega}_{\bot,1}$ of $1\bot$-sequences and a real function on $\mathbb{I}$ can be written as a program which inputs and outputs $1\bot$-sequences. In XPCF, one defines a function on $\Sigma^{\omega}_{\bot,1}$ only by specifying its behaviors for the cases that the first digit is 0 and 1. Then, its value for a sequence starting with a bottom is calculated by taking the meet of the values for the sequences obtained by filling the bottom with 0 and 1. The validity of the reduction rule of this calculus is justified by the adequacy theorem to a domain-theoretic semantics. Some example programs including addition and multiplication are shown. Expressive powers of XPCF and related languages are also investigated.

Keywords: Bottom, stream, real number computation, domain model, PCF, adequacy, parallel or

1 Introduction

Streams are a useful data structure used for expressing infinite sequences and one can implement real number computation with streams through signed digit expansion[1,2] or other expansions of real numbers[6]. However, since a stream can only be accessed one-way from left to right, if there is a bottom, i.e., a term whose evaluation does not terminate, in a stream, then a program get stuck when it tries to read in the value of the bottom cell and cannot input the rest of the sequence though it may contain valuable data.

Usually, a bottom is considered as a kind of programming error which should be avoided in a correct program. However, it is known that infinite sequences which may contain bottoms are useful in representing continuous topological spaces like $\mathbb{R}$. Here, we call an infinite sequence of $\Sigma \cup \{\bot\}$ which may contain at most one copy of bottom a $1\bot$-sequence. It is shown in [8] and [15] that $\mathbb{R}$ and $\mathbb{I} = [0, 1]$ can be

[1] Email: terayama@i.h.kyoto-u.ac.jp

[2] Email: tsuiki@i.h.kyoto-u.ac.jp

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.com/locate/entcs

topologically embedded in the space $\Sigma^{\omega}_{\perp,1}$ of $1\perp$-sequences of Σ for $\Sigma = \{0,1\}$ and this embedding is called the Gray embedding in [15]. The signed-digit expansion and other admissible representations of $\mathbb{R}$ turn out to be redundant in the sense that infinitely many reals each satisfy the property of being represented by infinitely many codes[4,17]. On the other hand, with the Gray embedding, a unique code can be assigned to each real number by extending the code space with at most one copy of $\perp$. This embedding result is extended in [16] to other topological spaces and it is shown that any n-dimensional separable metric space can be topologically embedded in the space $\Sigma^{\omega}_{\perp,n}$ of $n\perp$-sequences.

[8] expressed a $1\perp$-sequence as a function from $\mathsf{N}_{\perp}$ to $\{-1, 1, \perp\}$ and used the parallel if operator pif to access $1\perp$-sequences and showed that real number algorithms can be expressed in PCF + pif. In order to evaluate $\mathsf{pif}\ L\ M\ N$, one need to evaluate L, M, and N in parallel. Therefore, pif operator causes explosion of parallel computations and it seems difficult to implement it efficiently. Martin Escardó proposed Real PCF[6] which is an extension of PCF with real numbers. It is based on interval domains and a kind of parallel conditional operator is used.

On the other hand, [15] restricted the number of $\perp$ to one and introduced an IM2-machine (Indeterministic Multiheads Type2 Machine) which enables extended stream access to $1\perp$-sequences. However, the behavior of an IM2-machine needs to be specified through a set of overlapping rules and therefore functions expressible with IM2-machines are multi-valued functions in general. Moreover, a program of an IM2-machine is complicated because one needs to express its behaviors for inputs from extra heads.

In this paper, we introduce a calculus XPCF of $1\perp$-sequences with which one can express extended stream accesses to them. It is an extension of PCF with a datatype S of $1\perp$-sequences and is based on the algebraic domain $\mathbf{BD}$ of $1\perp$-sequences[16]. The datatype S has, in addition to the constructors $\mathsf{0} : \mathsf{S} \to \mathsf{S}$ and $\mathsf{1} : \mathsf{S} \to \mathsf{S}$ to prepend a digit to a sequence, constructors $\overline{\mathsf{0}} : \mathsf{S} \to \mathsf{S}$ and $\overline{\mathsf{1}} : \mathsf{S} \to \mathsf{S}$ to insert a digit as the second element of a sequence. However, a function on S is defined by expressing its behaviors only for cases the argument has the form $\mathsf{0}N$ and $\mathsf{1}N$ with the expression $\langle \mathsf{0}x \to M_0; \mathsf{1}x \to M_1\rangle$. It means a function on $\Sigma^{\omega}_{\perp,1}$ to apply $[\![\lambda x.M_0]\!]$ to s if the argument is $0s$, to apply $[\![\lambda x.M_1]\!]$ to s if the argument is $1s$, and apply both of them to s and take the meet of the results if the argument is $\perp s$. We designed a reduction rule and proved that this calculus has the computational adequacy property with respect to its domain-theoretic model. XPCF can be considered as an algebraic domain variant of Real PCF.

We give some example programs of XPCF including addition and multiplication on $\mathbb{I}$ through the Gray embedding. We also studied the expressive power of this language and showed that XPCF has the same expressive power as PCF + pif on types which do not contain S, all computable elements of $\mathbf{BD}$ are expressible on type S, and that if we extend XPCF with the $\exists$ operator, then all the computable elements in the semantic domains are expressible.

As [7] showed, any real number calculus which is adequate to the interval domain model and in which the average function can be represented, does not have a sequential reduction system. Their proof also applies to our model with some modifications and thus any sequential reduction system of this calculus is not ade-

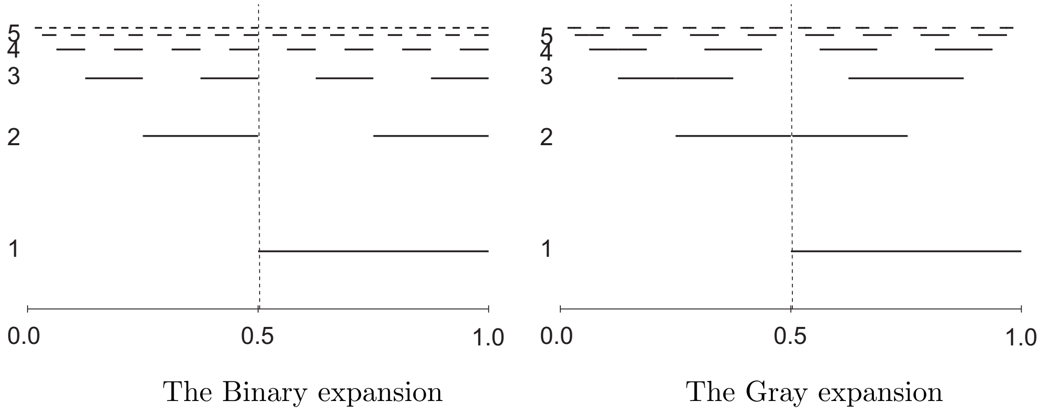


Fig. 1. The Binary and Gray expansion of $\mathbb{I}$

quate. We designed a sequential reduction strategy of XPCF and implemented it with Haskell. Though it is not adequate and some of the terms cannot be reduced to their denotations, it sequentially evaluates many of the terms like addition and multiplication.

In the next section, we start with explaining the Gray embedding of $\mathbb{I}$ in $\Sigma^{\omega}_{\perp,1}$ and introducing the domain **BD** of $1\perp$-sequences. In Section 3, we define the syntax and semantics of XPCF and, in Section 4, we show how real functions can be expressed in XPCF with some program examples. Then, we give reduction rules of XPCF in Section 5 and show the adequacy property in Section 6. In section 7, we study expressive powers of XPCF.

Notations: Recall that we fix $\Sigma = \{0, 1\}$. We denote by Γ^* the set of finite sequences of a character set Γ and by Γ^{ω} the set of infinite sequences of Γ. We define $\Gamma^{\infty} = \Gamma^* \cup \Gamma^{\omega}$, which is a Scott domain, i.e., a bounded complete ω-algebraic dcpo. Let $\Sigma_{\perp} = \Sigma \cup \{\perp\}$, and $\Sigma^{\omega}_{\perp}$ be the set of infinite sequences of $\Sigma_{\perp}$. $\Sigma_{\perp}$ has the order generated by $\perp \sqsubseteq 0$ and $\perp \sqsubseteq 1$. On $\Sigma^{\omega}_{\perp}$, we define the order $\sqsubseteq$ as $s \sqsubseteq t$ if $s(n) \sqsubseteq t(n)$ for every n. $(\Sigma^{\omega}_{\perp}, \sqsubseteq)$ is a Scott domain. We define $\Sigma^{\omega}_{\perp,1} = \{s \in \Sigma^{\omega}_{\perp} \mid s \text{ contains at most one } \perp\}$.

2 Real number computation and $1\perp$-sequences

2.1 Gray embedding

The Gray expansion is an expansion of $\mathbb{I}$ as infinite sequences of Σ which is different from the ordinary binary expansion [15]. It is based on Gray code[10], which is a coding of natural numbers with Σ different from the binary code. Figure 1 shows the binary and Gray expansion of $\mathbb{I}$. In the binary expansion of x, the head h of the expansion indicates whether x is in [0, 1/2] or in [1/2, 1] and the tail is the expansion of $f(x, h)$ for f the function defined as

$$f(x,h) = \begin{cases} 2x & (\text{if } h = 0) \\ 2x - 1 & (\text{if } h = 1) \end{cases}.$$

Thus, with the binary expansion, the tail of the expansion of 1/2 depends on the choice of the head character h and 1/2 has two expansions 1000... and 0111.... On the other hand, the head of the Gray expansion is the same as that of the binary expansion, whereas the tail is the expansion of $t(x)$ for t the so-called tent function:

$$t(x) = \begin{cases} 2x & (0 \leq x \leq 1/2) \\ 2(1-x) & (1/2 < x \leq 1) \end{cases}.$$

Note that $t(x)$ is continuous on $x = 1/2$ and therefore the tail of the expansion does not depend on the choice of the first digit. Actually, the two expansions of $1/2$ are 01000... and 11000... which coincide from the second character. It means that the value is half not depending on the first character. Therefore, we leave the first character undefined ($\perp$) and define a new expansion of $1/2$ as $\perp 1000\ldots$. It is also the case for expansions of dyadic numbers (rational numbers of the form $m/2^k$) and therefore we assign codes of the form $p\perp 1000\ldots$ for $p \in \{0,1\}^*$ to those numbers. In this way, we have a mapping $\varphi : \mathbb{I} \to \Sigma^\omega_{\perp,1}$ called the Gray-embedding as follows.

Definition 2.1 [[15]] Let $P : \mathbb{I} \to \Sigma_\perp$ be the map

$$P(x) = \begin{cases} 0 & (x < 1/2) \\ \perp & (x = 1/2) \\ 1 & (x > 1/2) \end{cases}.$$

the Gray embedding φ is a function from $\mathbb{I}$ to $\Sigma^\omega_{\perp,1}$ defined as $\varphi(x)(n) = P(t^n(x))$ $(n = 0, 1, \ldots)$.

An embedding of $\mathbb{R}$ in $\{-1,1\}^\omega_{\perp,1}$ is defined in [8] independently by Gianantonio, and the Gray embedding is essentially the same as the restriction of his embedding to $\mathbb{I}$. We call the $1\perp$-sequence $\varphi(x)$ the modified Gray expansion of x. The Gray embedding φ is actually a topological embedding with the topology of $\Sigma^\omega_{\perp,1}$ the subspace topology of the Scott topology of $\Sigma^\omega_\perp$.

2.2 Domains of 1⊥-sequences

We explain the domain **BD** in which terms of the type S of $1\perp$-sequences are interpreted [16]. Let $\Sigma^*_{\perp,1}$ be the set of finite $1\perp$-sequences of Σ. Here, $p \in {\Sigma_\perp}^*$ is a finite $1\perp$-sequence of Σ if $\perp$ appears at most once in p and $\perp$ is not the final character of p. We have $\Sigma^*_{\perp,1} = \{\epsilon, 0, 1, \perp 0, \perp 1, ...\}$ with ϵ the empty sequence. We can regard $\Sigma^*_{\perp,1}$ as a subset of $\Sigma^\omega_\perp$ by identifying $p \in \Sigma^*_{\perp,1}$ with $p\perp^\omega \in \Sigma^\omega_\perp$. We define $\mathbf{BD} = \Sigma^*_{\perp,1} \cup \Sigma^\omega_{\perp,1}$, which is a Scott subdomain of $\Sigma^\omega_\perp$ as Figure 2 shows. For $c \in \Sigma$, we also denote by c the continuous function from **BD** to **BD** to prepend c and denote by $\bar{c}$ the continuous function from **BD** to **BD** to insert c as the second character, where $\bar{c}(\epsilon)$ is defined as $\perp c$. We have the equation $\bar{b} \circ c = c \circ b$ for $b, c \in \Sigma$.

We consider that each finite sequence $s = d_0 d_1 \ldots d_{n-1}$ of $\{0, 1, \bar{0}, \bar{1}\}$ represents the element $d_0(d_1(\ldots(d_{n-1}(\epsilon))))$ of $\Sigma^*_{\perp,1}$ and each infinite sequence $s = d_0 d_1 \ldots$ of $\{0, 1, \bar{0}, \bar{1}\}$ represents the limit of the infinite increasing sequence $(s_n)_{n=0,1,\ldots}$ in **BD** for $s_n = d_0(d_1(\ldots(d_{n-1}(\epsilon))))$. Note that this limit exists in $\Sigma^\omega_{\perp,1}$. In particular, the sequence $b_0 b_1 \ldots b_{m-1} \overline{c_0 c_1} \ldots \overline{c_{n-1}}$ represents $b_0 b_1 \ldots b_{m-1} \perp c_0 c_1 \ldots c_{n-1} \in \Sigma^*_{\perp,1}$

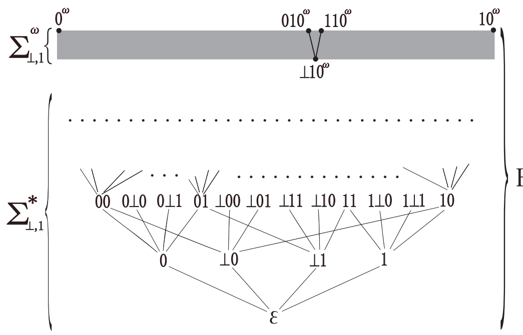


Fig. 2. The domain **BD**

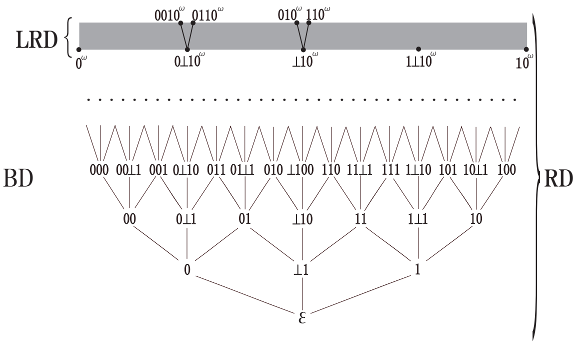


Fig. 3. The domain **RD**

(or $b_0 b_1 \dots b_{m-1}$ if $n = 0$), and the infinite sequence $b_0 b_1 \dots b_{m-1} \overline{c_0 c_1} \dots$ represents $b_0 b_1 \dots b_{m-1} \perp c_0 c_1 \dots \in \Sigma^{\omega}_{\perp,1}$.

Since **BD** is a Scott domain, the meet (i.e., the greatest lower bound) exists for any subset of **BD**. We show it explicitly, because it plays an important role in the semantics of XPCF. First, the meet on $\Sigma_{\perp} = \{0, 1, \perp\}$ is obviously defined. It is naturally extended to the meet $s \sqcap_{\Sigma^{\omega}_{\perp}} t$ in $\Sigma^{\omega}_{\perp}$ as $(s \sqcap_{\Sigma^{\omega}_{\perp}} t)(n) = s(n) \sqcap t(n)$. Let trunc be the function from $\Sigma^{\omega}_{\perp}$ to **BD** to truncate the sequence after the second $\perp$ to form a finite $1\perp$-sequence if it contains more than one copies of $\perp$, and returns itself if it does not.

Proposition 2.2 *The meet $s \sqcap t$ of $s, t \in$ **BD** is equal to* $\mathrm{trunc}(s \sqcap_{\Sigma^{\omega}_{\perp}} t)$.

We define the subdomain **RD** of **BD** which is used for expressing $\mathbb{I}$ through the Gray representation. We define

$$\mathbf{RD} = \{p\perp 10^{n} \ : \ p \in \Sigma^{*}, n \in \{0, 1, \dots, \omega\}\} \cup \Sigma^{\infty}.$$

It is a Scott domain. Let **LRD** be the subset $\{p\perp 10^{\omega} : p \in \Sigma^{*}\} \cup \Sigma^{\omega}$ of $\Sigma^{\omega}_{\perp,1}$. **LRD** is the set of limit (i.e., non-compact) elements of **RD** as Figure 3 shows. **LRD** consists of $\varphi(\mathbb{I})$ and those sequences obtained by filling a bottom of $s \in \varphi(\mathbb{I})$ with 0 and 1. One can see that $\mathbb{I}$ is a retract of **LRD** and $\mathbb{I}$ is homeomorphic to the set of minimal elements of **LRD** with the retract map $\delta : \mathbf{LRD} \to \mathbb{I}$ defined as $\delta(s) = x$ if $\varphi(x) \sqsubseteq s$. One can see that the triple $(\mathbf{RD}, \mathbf{LRD}, \delta)$ is a retract domain representation of $\mathbb{I}$ in the sense of [3] and we call the map $\delta : \mathbf{LRD} \to \mathbb{I}$ the Gray representation.

We can consider two codings of $\mathbb{I}$ based on the Gray embedding. The first one is obtained by identifying x with $\varphi(x)$ through the embedding and the other one is the Gray representation δ. For example, for $1/2 \in \mathbb{I}$ there are three codes $\perp 10^{\omega}, 010^{\omega}$ and 110^{ω} with respect to δ. On the other hand, $\perp 10^{\omega}$ is the unique codes for $1/2$ with φ. Based on these codings, we have two notions that a function on **BD** realize a function on $\mathbb{I}$.

Definition 2.3 Let F be a function from $\mathbf{BD}^{n}$ to **BD** and f be a (partial) real function from $\mathbb{I}^{n}$ to $\mathbb{I}$.

(1) F *exactly realizes* f if, for every $(x_1, \ldots, x_n) \in \mathrm{dom}(f)$,

$$F(\varphi(x_1), \ldots, \varphi(x_n)) = \varphi(f(x_1, \ldots, x_n)).$$

(2) F *realizes* f if, for every $(p_1, \ldots, p_n) \in (\delta^n)^{-1}(\mathrm{dom}(f))$,

$$\delta(F(p_1, \ldots, p_n)) = f(\delta(p_1), \ldots, \delta(p_n)).$$

3 Syntax and denotational semantics of XPCF

Throughout this paper, we write types and constants of syntactic entities in $\mathsf{Sans\text{-}serif}$ font and elements in domains in Roman font. Program names and the names of semantic domains are written in **Bold** font.

3.1 PCF

For the reader's convenience we review the syntax, the semantics, and the reduction rules of the language PCF in Table 1. See [11] or some textbooks like [14] for the details of PCF. For a term M, $FV(M)$ denotes the free variables of M and M is *closed* if $FV(M)$ is empty. A program is a closed term of a ground type. An *environment* ρ is a type-respecting map from the set of variables to $\bigcup\{D_\sigma | \sigma \,\text{type}\}$ and, for $a \in D_\sigma$, $\rho[a/x^\sigma]$ is the environment which maps x^σ to a and any other variable y^σ to $\rho(y^\sigma)$. If M is a closed term, then $[\![M]\!](\rho)$ does not depend on ρ and we write $[\![M]\!]$ for $[\![M]\!](\rho)$.

The operational semantics of PCF is given by the *immediate reduction relation* in Table 1. Evaluation of a program M is a constant c defined as

$$\mathrm{Eval}(M) = c \;\text{ iff }\; M \rhd^* c.$$

The following theorem is often referred to as the *Adequacy Property* of PCF. It asserts that the operational and denotational semantics coincide.

Theorem 3.1 *([11, Theorem 3.1]). For any PCF program M and constant c,*

$$\mathrm{Eval}(M) = c \;\text{ iff }\; [\![M]\!] = [\![c]\!].$$

3.2 Syntax and semantics of XPCF

The syntax and denotational semantics of XPCF is listed in Table 2. We list only the differences compared with the PCF specification. It has a ground type S such that $D_{\mathsf{S}} = \mathbf{BD}$ with constants $\mathsf{0}, \mathsf{1}, \overline{\mathsf{0}}, \overline{\mathsf{1}}$ of type $\mathsf{S} \to \mathsf{S}$ which denote the functions $0, 1, \overline{0}, \overline{1}$, respectively. For a variable of type S, we omit the type and write x for x^{S}, for simplicity. We have function terms $\langle \mathsf{0}x \to M_0; \mathsf{1}x \to M_1\rangle$ and $\langle\!\langle \mathsf{0}x \to M_0; \mathsf{1}x \to M_1\rangle\!\rangle$ of type $\mathsf{S} \to \sigma$ for M_0 and M_1 terms of type σ. The variable x is a bound variable of $\langle \mathsf{0}x \to M_0; \mathsf{1}x \to M_1\rangle$ and $\langle\!\langle \mathsf{0}x \to M_0; \mathsf{1}x \to M_1\rangle\!\rangle$.

We call $\langle \mathsf{0}x \to M_0; \mathsf{1}x \to M_1\rangle$ and $\langle\!\langle \mathsf{0}x \to M_0; \mathsf{1}x \to M_1\rangle\!\rangle$ *extended conditional terms*. For the two functions $f_0 = [\![\lambda x.M_0]\!]$ and $f_1 = [\![\lambda x.M_1]\!]$ from D_{S} to D_σ, the function $f = [\![\langle \mathsf{0}x^{\mathsf{S}} \to M_0; \mathsf{1}x^{\mathsf{S}} \to M_1\rangle]\!] : D_{\mathsf{S}} \to D_\sigma$ returns $f_0(s)$ if the argument is

Syntax of PCF

Types: $\sigma ::= \mathsf{B} \,|\, \mathsf{N} \,|\, \sigma \to \sigma$

Variables(of type σ): $x^\sigma ::= x^\sigma, y^\sigma, z^\sigma, \ldots$

Constants: $c ::= \mathsf{tt}, \mathsf{ff}, \mathsf{if}_\sigma, \mathsf{Y}_\tau, \mathsf{k}_n, \mathsf{inc}, \mathsf{dec}, \mathsf{zero}$ (σ:ground type, τ:type, $n \in \mathbb{N}$)

Terms: $M ::= x^\tau \,|\, c \,|\, (MM) \,|\, (\lambda x^\sigma.M)$

Typing Rules:

$$x^\sigma : \sigma \qquad \mathsf{tt} : \mathsf{B} \qquad \mathsf{ff} : \mathsf{B} \qquad \mathsf{k}_n : \mathsf{N} \qquad \mathsf{inc} : \mathsf{N} \to \mathsf{N}$$

$$\mathsf{dec} : \mathsf{N} \to \mathsf{N} \qquad \mathsf{zero} : \mathsf{N} \to \mathsf{B} \qquad \mathsf{if}_\sigma : \mathsf{B} \to \sigma \to \sigma \to \sigma$$

$$\mathsf{Y}_\sigma : (\sigma \to \sigma) \to \sigma \qquad \frac{M : \tau}{\lambda x^\sigma.M : \sigma \to \tau} \qquad \frac{M : \sigma \to \tau \quad N : \sigma}{MN : \tau}$$

Denotational semantics of PCF

Domains: D_B :the flat domain $\{\bot_\mathsf{B}, tt, ff\}$ of truth values.
D_N :the flat domain $\{\bot_\mathsf{N}, 0, 1, \ldots\}$ of natural numbers.
$D_{\sigma\to\tau} := [D_\sigma \to D_\tau]$.

Interpretation of constants:

$$[\![\mathsf{tt}]\!] = tt \qquad [\![\mathsf{ff}]\!] = ff \qquad [\![\mathsf{k}_n]\!] = n$$

$$[\![\mathsf{inc}]\!] = \lambda n \in D_\mathsf{N}. \begin{cases} n+1 & (n \neq \bot_\mathsf{N}) \\ \bot_\mathsf{N} & (n = \bot_\mathsf{N}) \end{cases} \qquad [\![\mathsf{dec}]\!] = \lambda n \in D_\mathsf{N}. \begin{cases} n-1 & (n \geq 1) \\ \bot_\mathsf{N} & (n \in \{\bot_\mathsf{N}, 0\}) \end{cases}$$

$$[\![\mathsf{zero}]\!] = \lambda n \in D_\mathsf{N}. \begin{cases} tt & (n = 0) \\ ff & (n > 0) \\ \bot_\mathsf{B} & (n = \bot_\mathsf{N}) \end{cases} \qquad [\![\mathsf{Y}_\sigma]\!] = \lambda F \in D_{\sigma\to\sigma}. \textstyle\bigsqcup_{n\in\mathsf{N}} F^n(\bot_\sigma)$$

$$[\![\mathsf{if}_\sigma]\!] = \lambda b \in D_\mathsf{B}.\lambda x \in D_\sigma.\lambda y \in D_\sigma. \begin{cases} x & (b = tt) \\ y & (b = ff) \\ \bot_\sigma & (b = \bot_\mathsf{B}) \end{cases}$$

Denotational semantics:

(i) $[\![x^\sigma]\!](\rho) = \rho(x^\sigma)$ (ii) $[\![c]\!](\rho) = [\![c]\!]$

(iii) $[\![MN]\!](\rho) = [\![M]\!](\rho)([\![N]\!](\rho))$ (iv) $[\![\lambda x^\sigma.M]\!](\rho) = \lambda a \in D_\sigma.[\![M]\!](\rho[a/x^\sigma])$

Operational semantics of PCF

Reduction rules:

$$(\lambda x^\sigma.M)N \rhd M[N/x^\sigma] \qquad \mathsf{Y}_\sigma\, M \rhd M(\mathsf{Y}_\sigma\, M) \qquad \mathsf{inc}\,\mathsf{k}_n \rhd \mathsf{k}_{n+1} \qquad \mathsf{dec}\,\mathsf{k}_{n+1} \rhd \mathsf{k}_n$$

$$\mathsf{zero}\,\mathsf{k}_0 \rhd \mathsf{tt} \qquad \mathsf{zero}\,\mathsf{k}_{n+1} \rhd \mathsf{ff} \qquad \mathsf{if}_\sigma\,\mathsf{tt}\, M\, N \rhd M \qquad \mathsf{if}_\sigma\,\mathsf{ff}\, M\, N \rhd N$$

$$\text{(APP-L)}\ \frac{M \rhd M'}{M\, N \rhd M'\, N} \qquad \frac{N \rhd N'}{M\, N \rhd M\, N'}\ (\text{if } M \text{ is } \mathsf{if}_\sigma, \mathsf{inc}, \mathsf{dec} \text{ or } \mathsf{zero})$$

Table 1
Syntax and semantics of PCF

$0s$ and $f_1(s)$ if the argument is $1s$. For the case the argument starts with $\bot$, we define $f(\bot s) = f_0(s) \sqcap f_1(s)$, which is the meet of $f_0(s)$ and $f_1(s)$ in D_σ. Here, meets on D_N and D_B are obviously defined, meets on D_S are explained in Section 2.2, and the meet of two functions $g, h \in D_{\sigma\to\tau}$ is the pointwise meet function $(g \sqcap h)(x) = g(x) \sqcap h(x)$. We define $f(\epsilon) = \bot_\sigma$. Thus, f is a strict function. It means that we adopt call by value semantics to an application of $\langle 0x^\mathsf{S} \to M_0; 1x^\mathsf{S} \to M_1 \rangle$.

The meaning of the term $\langle\!\langle 0x^\mathsf{S} \to M_0; 1x^\mathsf{S} \to M_1 \rangle\!\rangle$ is different from that of $\langle 0x^\mathsf{S} \to$

Syntax of XPCF Syntax of PCF extended with the followings.

Types: S

Constants: $\mathsf{0}, \mathsf{1}, \bar{\mathsf{0}}, \bar{\mathsf{1}}$

Terms: $\langle \mathsf{0}x^{\mathsf{S}} \to M; \mathsf{1}x^{\mathsf{S}} \to M \rangle \mid \langle\!\langle \mathsf{0}x^{\mathsf{S}} \to M; \mathsf{1}x^{\mathsf{S}} \to M \rangle\!\rangle$

Typing Rules:

$$\mathsf{0} : \mathsf{S} \to \mathsf{S} \qquad \mathsf{1} : \mathsf{S} \to \mathsf{S} \qquad \bar{\mathsf{0}} : \mathsf{S} \to \mathsf{S} \qquad \mathsf{1} : \mathsf{S} \to \mathsf{S}$$

$$\frac{M_0 : \sigma \quad M_1 : \sigma}{\langle \mathsf{0}x^{\mathsf{S}} \to M_0; \mathsf{1}x^{\mathsf{S}} \to M_1 \rangle : \mathsf{S} \to \sigma} \qquad \frac{M_0 : \sigma \quad M_1 : \sigma}{\langle\!\langle \mathsf{0}x^{\mathsf{S}} \to M_0; \mathsf{1}x^{\mathsf{S}} \to M_1 \rangle\!\rangle : \mathsf{S} \to \sigma}$$

Denotational semantics of XPCF Semantics of PCF extended with the followings.

Domains: $D_{\mathsf{S}} = \Sigma^{\infty}_{\perp,1}$

Interpretation of constants:

$[\![\mathsf{0}]\!] = 0 \in D_{\mathsf{S} \to \mathsf{S}}$ (where $0(s) = 0s$ for $s \in D_{\mathsf{S}}$)

$[\![\mathsf{1}]\!] = 1 \in D_{\mathsf{S} \to \mathsf{S}}$ (where $1(s) = 1s$ for $s \in D_{\mathsf{S}}$)

$[\![\bar{\mathsf{0}}]\!] = \bar{0} \in D_{\mathsf{S} \to \mathsf{S}}$ (where $\bar{0}(as) = a0s$ for $as \in D_{\mathsf{S}}$, $\bar{0}(\epsilon) = \perp 0$)

$[\![\bar{\mathsf{1}}]\!] = \bar{1} \in D_{\mathsf{S} \to \mathsf{S}}$ (where $\bar{1}(as) = a1s$ for $as \in D_{\mathsf{S}}$, $\bar{1}(\epsilon) = \perp 1$)

Denotational semantics:

(v) $[\![\langle \mathsf{0}x^{\mathsf{S}} \to M_0; \mathsf{1}x^{\mathsf{S}} \to M_1 \rangle]\!](\rho) =$

$$\lambda s \in D_{\mathsf{S}}. \begin{cases} \perp_\sigma & (\text{if } s = \epsilon) \\ [\![M_0]\!](\rho[s'/x^{\mathsf{S}}]) & (\text{if } s = 0s') \\ [\![M_1]\!](\rho[s'/x^{\mathsf{S}}]) & (\text{if } s = 1s') \\ [\![M_0]\!](\rho[s'/x^{\mathsf{S}}]) \sqcap [\![M_1]\!](\rho[s'/x^{\mathsf{S}}]) & (\text{if } s = \perp s') \end{cases}$$

(vi) $[\![\langle\!\langle \mathsf{0}x^{\mathsf{S}} \to M_0; \mathsf{1}x^{\mathsf{S}} \to M_1 \rangle\!\rangle]\!](\rho) =$

$$\lambda s \in D_{\mathsf{S}}. \begin{cases} [\![M_0]\!](\rho[\epsilon/x^{\mathsf{S}}]) \sqcap [\![M_1]\!](\rho[\epsilon/x^{\mathsf{S}}]) & (\text{if } s = \epsilon) \\ [\![M_0]\!](\rho[s'/x^{\mathsf{S}}]) & (\text{if } s = 0s') \\ [\![M_1]\!](\rho[s'/x^{\mathsf{S}}]) & (\text{if } s = 1s') \\ [\![M_0]\!](\rho[s'/x^{\mathsf{S}}]) \sqcap [\![M_1]\!](\rho[s'/x^{\mathsf{S}}]) & (\text{if } s = \perp s') \end{cases}$$

Table 2
Syntax and denotational semantics of XPCF

$M_0; \mathsf{1}x^{\mathsf{S}} \to M_1 \rangle$ only for the case of ϵ, and $[\![\langle\!\langle \mathsf{0}x^{\mathsf{S}} \to M_0; \mathsf{1}x^{\mathsf{S}} \to M_1 \rangle\!\rangle]\!]$ is not a strict function. Note that if we identify ϵ and $\perp^\omega$ and match $\perp s'$ with $\perp^\omega$, then the last case of the semantics of $\langle\!\langle \mathsf{0}x^{\mathsf{S}} \to M_0; \mathsf{1}x^{\mathsf{S}} \to M_1 \rangle\!\rangle$ subsumes the first case. Note that both functions $[\![\langle \mathsf{0}x^{\mathsf{S}} \to M_0; \mathsf{1}x^{\mathsf{S}} \to M_1 \rangle]\!]$ and $[\![\langle\!\langle \mathsf{0}x^{\mathsf{S}} \to M_0; \mathsf{1}x^{\mathsf{S}} \to M_1 \rangle\!\rangle]\!]$ are continuous. Our intention in introducing two kinds of extended conditional terms is that $\langle \mathsf{0}x^{\mathsf{S}} \to M_0; \mathsf{1}x^{\mathsf{S}} \to M_1 \rangle$ is used in writing a program and $\langle\!\langle \mathsf{0}x^{\mathsf{S}} \to M_0; \mathsf{1}x^{\mathsf{S}} \to M_1 \rangle\!\rangle$ is used only in reduction steps, which we explain in Section 5. We call a closed ground type term a program if it does not contain extended conditional terms of the form $\langle\!\langle \mathsf{0}x^{\mathsf{S}} \to M_0; \mathsf{1}x^{\mathsf{S}} \to M_1 \rangle\!\rangle$ as subterms.

4 Program examples of XPCF

The function **nh** to invert the first digit is written as

$$\mathbf{nh} = \langle 0x \to 1x; 1x \to 0x \rangle.$$

Note that $[\![\mathbf{nh}]\!](\bot s) = 0s \sqcap 1s = \bot s$ for $s \in \Sigma^\omega$.

The function **ns** to invert the second digit is written as

$$\mathbf{ns} = \langle 0x \to 0(\mathbf{nh}\ x); 1x \to 1(\mathbf{nh}\ x) \rangle.$$

The following terms **head** : $\mathsf{S} \to \mathsf{B}$ and **tail** : $\mathsf{S} \to \mathsf{S}$ are the head and the tail function on D_{S}.

$$\mathbf{head} = \langle 0x \to \mathsf{ff}; 1x \to \mathsf{tt} \rangle,$$
$$\mathbf{tail} = \langle 0x \to x; 1x \to x \rangle.$$

Here, we identify $0, 1, \bot \in \Sigma_\bot$ with $\mathit{ff}, \mathit{tt}, \bot_{\mathsf{B}} \in D_{\mathsf{B}}$, respectively. Note that there is no cons function: $\mathsf{B} \to \mathsf{S} \to \mathsf{S}$ because if we prepend $\bot$ to a $1\bot$-sequence, then the result may not be a $1\bot$-sequence. The function **inv** to invert all the digits is written as

$$\mathbf{inv} = \mathsf{Y}_{\mathsf{S}\to\mathsf{S}}(\lambda f^{\mathsf{S}\to\mathsf{S}}.\langle 0x \to 1(f\ x); 1x \to 0(f\ x) \rangle).$$

For simplicity, we use the recursive definition notation to abbreviate a term defined with the Y operator. For example, **inv** is written as

$$\mathbf{inv} = \langle 0x \to 1(\mathbf{inv}\ x); 1x \to 0(\mathbf{inv}\ x) \rangle.$$

We show how real numbers and real functions are expressed in XPCF. Since $\varphi(0) = 0^\omega$, $\varphi(1) = 10^\omega$ and $\varphi(1/2) = \bot 10^\omega$, we can express these numbers as

$$\mathbf{0} = Y_{\mathsf{S}}\ 0,$$
$$\mathbf{1} = 1(\mathsf{Y}_{\mathsf{S}}\ 0),$$
$$\mathbf{1/2} = \bar{1}(\mathsf{Y}_{\mathsf{S}}\ \bar{0}).$$

In Section 2.1, we defined notions that a function on **BD** (exactly) realizes a function on $\mathbb{I}$. We say that a closed XPCF term (exactly) realizes a real function if it denotes a function which (exactly) realizes the function. The program

$$\mathbf{div2} = \lambda x^{\mathsf{S}}.0x.$$

realizes the function $\mathrm{div2}(x) = x/2$ but does not exactly realize it because $[\![\mathbf{div2}]\!](10^\omega) = 010^\omega$ whereas $\varphi(1/2) = \bot 10^\omega$. There is also a program which exactly realizes div2, which is given later. Since the complement function $\mathrm{comp}(x) = 1 - x$ is realized by the function to invert the first digit, comp is exactly realized by the program **nh**. The tent function t is exactly realized by **tail**.

Programs of addition (average) **av**, subtraction **sub**, multiplication **mult** and **div2b** which exactly realizes div2 can be written as follows.

$$\begin{aligned}\mathbf{av} = \langle & 0x \to \langle 0y \to 0(\mathbf{av}\,x\,y); 1y \to \bar{1}(\mathbf{ns}(\mathbf{av}\,x\,(\mathbf{nh}\,y))) \rangle; \\ & 1x \to \langle 0y \to \bar{1}(\mathbf{ns}(\mathbf{av}\,(\mathbf{nh}\,x)\,y)); 1y \to 1(\mathbf{av}\,x\,y) \rangle \rangle\end{aligned}$$

$$\begin{aligned}\mathbf{sub} = \langle & 0x \to \langle 0y \to 0(\mathbf{sub}\,x\,y); 1y \to \mathsf{Y}_{\mathsf{S}}0 \rangle; \\ & 1x \to \langle 0y \to \mathbf{nh}(\mathbf{av}\,x\,y); 1y \to 0(\mathbf{sub}\,y\,x) \rangle \rangle\end{aligned}$$

$$\begin{aligned}\mathbf{mult} = \langle & 0x \to \langle 0y \to 0(0(\mathbf{mult}\,x\,y)); 1y \to 0(\mathbf{mult}\,x\,(1y)) \rangle; \\ & 1x \to \langle 0y \to 0(\mathbf{mult}\,(1x)\,y); \\ & \qquad 1y \to \mathbf{av}\,(\mathbf{nh}(\mathbf{av}\,x\,y))\,(1(\mathbf{nh}(\mathbf{mult}\,(\mathbf{nh}\,x)\,(\mathbf{nh}\,y)))) \rangle \rangle\end{aligned}$$

Reduction rule of XPCF

In addition to the reduction rule of PCF, we have the following rules.

(COND $\mathsf{0}$) $\langle \mathsf{0}x \to M_0; \mathsf{1}x \to M_1 \rangle (\mathsf{0}N) \triangleright M_0[N/x]$

$\langle\!\langle \mathsf{0}x \to M_0; \mathsf{1}x \to M_1 \rangle\!\rangle (\mathsf{0}N) \triangleright M_0[N/x]$

(COND $\mathsf{1}$) $\langle \mathsf{0}x \to M_0; \mathsf{1}x \to M_1 \rangle (\mathsf{1}N) \triangleright M_1[N/x]$

$\langle\!\langle \mathsf{0}x \to M_0; \mathsf{1}x \to M_1 \rangle\!\rangle (\mathsf{1}N) \triangleright M_1[N/x]$

(COND $\bar{\mathsf{0}}$) $\langle \mathsf{0}x \to M_0; \mathsf{1}x \to M_1 \rangle (\bar{\mathsf{0}}N) \triangleright \langle\!\langle \mathsf{0}x \to M_0[\mathsf{0}x/x]; \mathsf{1}x \to M_1[\mathsf{0}x/x] \rangle\!\rangle N$

$\langle\!\langle \mathsf{0}x \to M_0; \mathsf{1}x \to M_1 \rangle\!\rangle (\bar{\mathsf{0}}N) \triangleright \langle\!\langle \mathsf{0}x \to M_0[\mathsf{0}x/x]; \mathsf{1}x \to M_1[\mathsf{0}x/x] \rangle\!\rangle N$

(COND $\bar{\mathsf{1}}$) $\langle \mathsf{0}x \to M_0; \mathsf{1}x \to M_1 \rangle (\bar{\mathsf{1}}N) \triangleright \langle\!\langle \mathsf{0}x \to M_0[\mathsf{1}x/x]; \mathsf{1}x \to M_1[\mathsf{1}x/x] \rangle\!\rangle N$

$\langle\!\langle \mathsf{0}x \to M_0; \mathsf{1}x \to M_1 \rangle\!\rangle (\bar{\mathsf{1}}N) \triangleright \langle\!\langle \mathsf{0}x \to M_0[\mathsf{1}x/x]; \mathsf{1}x \to M_1[\mathsf{1}x/x] \rangle\!\rangle N$

(OUT 1) $\langle\!\langle \mathsf{0}x \to \mathsf{d}M_0; \mathsf{1}x \to \mathsf{d}M_1 \rangle\!\rangle N \triangleright \mathsf{d}(\langle\!\langle \mathsf{0}x \to M_0; \mathsf{1}x \to M_1 \rangle\!\rangle N) \quad (\mathsf{d} \in \{\mathsf{0}, \mathsf{1}, \bar{\mathsf{0}}, \bar{\mathsf{1}}\})$

(OUT 2) $\langle\!\langle \mathsf{0}x \to \mathsf{b}(\mathsf{c}M_0); \mathsf{1}x \to \mathsf{b}'(\mathsf{c}M_1) \rangle\!\rangle N \triangleright \bar{\mathsf{c}}(\langle\!\langle \mathsf{0}x \to \mathsf{b}M_0; \mathsf{1}x \to \mathsf{b}'M_1 \rangle\!\rangle N) \ (\mathsf{b} \neq \mathsf{b}')$

(OUT 3) $\langle\!\langle \mathsf{0}x \to \bar{\mathsf{b}}M_0; \mathsf{1}x \to \mathsf{c}(\mathsf{b}M_1) \rangle\!\rangle N \triangleright \bar{\mathsf{b}}(\langle\!\langle \mathsf{0}x \to M_0; \mathsf{1}x \to \mathsf{c}M_1 \rangle\!\rangle N) \ (\mathsf{b}, \mathsf{c} \in \{\mathsf{0}, \mathsf{1}\})$

(OUT 4) $\langle\!\langle \mathsf{0}x \to \mathsf{c}(\mathsf{b}M_0); \mathsf{1}x \to \bar{\mathsf{b}}M_1 \rangle\!\rangle N \triangleright \bar{\mathsf{b}}(\langle\!\langle \mathsf{0}x \to \mathsf{c}M_0; \mathsf{1}x \to M_1 \rangle\!\rangle N) \ (\mathsf{b}, \mathsf{c} \in \{\mathsf{0}, \mathsf{1}\})$

(OUT 5) $\langle\!\langle \mathsf{0}x \to c; \mathsf{1}x \to c \rangle\!\rangle \triangleright c \quad (c \in \{\mathsf{tt}, \mathsf{ff}, \mathsf{k}_n\})$

(BAR) $\bar{\mathsf{b}}(\mathsf{c}M) \triangleright \mathsf{c}(\mathsf{b}M) \quad (\mathsf{b}, \mathsf{c} \in \{\mathsf{0}, \mathsf{1}\})$

(PERM) $\langle \mathsf{0}x \to M_0; \mathsf{1}x \to M_1 \rangle NL \triangleright \langle \mathsf{0}x \to M_0L; \mathsf{1}x \to M_1L \rangle N$

$\langle\!\langle \mathsf{0}x \to M_0; \mathsf{1}x \to M_1 \rangle\!\rangle NL \triangleright \langle\!\langle \mathsf{0}x \to M_0L; \mathsf{1}x \to M_1L \rangle\!\rangle N$

(If $x \in FV(L)$, then rename the bound variable x to avoid variable collision.)

(LEFT) $\dfrac{M_0 \triangleright M_0'}{\langle\!\langle \mathsf{0}x \to M_0; \mathsf{1}x \to M_1 \rangle\!\rangle \triangleright \langle\!\langle \mathsf{0}x \to M_0'; \mathsf{1}x \to M_1 \rangle\!\rangle}$

(RIGHT) $\dfrac{M_1 \triangleright M_1'}{\langle\!\langle \mathsf{0}x \to M_0; \mathsf{1}x \to M_1 \rangle\!\rangle \triangleright \langle\!\langle \mathsf{0}x \to M_0; \mathsf{1}x \to M_1' \rangle\!\rangle}$

(APP-R) $\dfrac{N \triangleright N'}{MN \triangleright MN'}$ (if M is $\mathsf{0}, \mathsf{1}, \bar{\mathsf{0}}, \bar{\mathsf{1}}, \langle \mathsf{0}x \to M_0; \mathsf{1}x \to M_1 \rangle$ or $\langle\!\langle \mathsf{0}x \to M_0; \mathsf{1}x \to M_1 \rangle\!\rangle$)

Table 3
Operational semantics of XPCF

$\mathbf{div2b} = \langle \mathsf{0}x \to \mathsf{0}(\mathsf{0}x); \mathsf{1}x \to \bar{\mathsf{1}}(\mathbf{f}x) \rangle$
$\mathbf{f} = \langle \mathsf{0}x \to \bar{\mathsf{0}}(\mathbf{f}x); \mathsf{1}x \to \mathsf{0}(\mathsf{1}x) \rangle$

Here, $[\![\mathbf{f}]\!]$ is a function which satisfies $[\![\mathbf{f}]\!](0^\omega) = \perp 0^\omega$ and $[\![\mathbf{f}]\!](x) = 0x$ if x contains the character 1.

5 Operational semantics of XPCF

5.1 Operational semantics of XPCF

Table 3 shows the reduction rule of XPCF. For $\mathsf{d} \in \{\mathsf{0}, \mathsf{1}, \bar{\mathsf{0}}, \bar{\mathsf{1}}\}$, we say that a term M of type S outputs d if M is reduced to $\mathsf{d}M'$.

We explain how the reduction of a term $\langle \mathsf{0}x \to M_0; \mathsf{1}x \to M_1 \rangle\ N$ proceeds. The first lines of rules (COND $\mathsf{0}$), (COND $\mathsf{1}$), (COND $\bar{\mathsf{0}}$), and (COND $\bar{\mathsf{1}}$) are for the reduction of an application term $\langle \mathsf{0}x \to M_0; \mathsf{1}x \to M_1 \rangle\ N$. Note that a closed term N is reduced by (APP-R) to one of these four forms if $[\![N]\!]$ is not $\perp$ by the adequacy theorem in the next section. If N has the form $\overline{0}N'$, (COND $\overline{0}$) is applied and then we have a term $M_0[0x/x]$ and $M_1[1x/x]$. After that, M_0 and M_1 are evaluated by (LEFT) and (RIGHT) only with the additional information that the first character of x is 0. Note that if M_0 contains x, then $M_0[0x/x]$ also contains x and therefore it is expected that this evaluation terminates when it requires the value of x. Then, (BAR) rule is used to arrange outputs of $M_0[0x/x]$ and $M_1[0x/x]$ to the form $b_0b_1 \dots b_k\overline{c_0c_1}\dots\overline{c_j}$ for $b_i, c_i \in \{0, 1\}$. After that, if they coincide on the first or the second digit, then it makes an output with rules (OUT 1) to (OUT 5) and repeat it until no more output is possible. Thus, we obtain a term of the form $d_0d_1 \dots d_i(\langle\!\langle 0x \to M_0'; 1x \to M_1' \rangle\!\rangle\ N')$ and we can continue this process to the subterm $\langle\!\langle 0x \to M_0'; 1x \to M_1' \rangle\!\rangle\ N'$ with (APP-R) since all the rules applicable to $\langle 0x \to M_0; 1x \to M_1 \rangle\ N$ are also applicable to terms of the form $\langle\!\langle 0x \to M_0; 1x \to M_1 \rangle\!\rangle\ N$.

One can see that the above reduction procedure fails to reduce $\langle 0x \to M_0; 1x \to M_1 \rangle\ N\ L$ because the output of L cannot be fed to function terms M_0 and M_1. For the evaluation of this term, we need the (PERM) rule. Suppose that $\langle 0x \to M_0; 1x \to M_1 \rangle : \mathsf{S} \to \mathsf{S} \to \mathsf{S}$ and M_0 and M_1 are extended conditional terms of the form $\langle 0y \to ...; 1y \to ... \rangle$. We first reduce the term $\langle 0x \to M_0; 1x \to M_1 \rangle\ N\ L$ to $\langle 0x \to (M_0\ L); 1x \to (M_1\ L) \rangle\ N$ with the (PERM) rule and then reduce it as we explained. The (PERM) rule corresponds to reducing the lambda term $(\lambda x.M)\ N\ L$ to $(\lambda x.(M\ L))\ N$, and it is similar to the permutative conversion rule used for proof normalization in proof theory [12].

One may wonder why we distinguish $\langle 0x \to M_0; 1x \to M_1 \rangle$ with $\langle\!\langle 0x \to M_0; 1x \to M_1 \rangle\!\rangle$ because we can make the same reduction if we replace the former with the latter. However, strictness of $\langle 0x \to M_0; 1x \to M_1 \rangle$ plays an important role in writing recursively defined functions. Many of the functions on S are defined with the Y operator as $F = \mathsf{Y}_{\mathsf{S}\to\mathsf{S}}(\lambda f.M)$ with $M = \langle 0x \to M_0; 1x \to M_1 \rangle$ and it is reduced to $M[\mathsf{Y}_{\mathsf{S}\to\mathsf{S}}(\lambda f.M)/f]$ which cannot be reduced any more. If $M = \langle\!\langle 0x \to M_0; 1x \to M_1 \rangle\!\rangle$ instead, then copies of $\mathsf{Y}_{\mathsf{S}\to\mathsf{S}}(\lambda f.M)$ in M_0 and M_1 can be reduced and therefore it causes an infinite computation even if no argument is given to the function.

5.2 A sequential strategy of XPCF

Though one needs to evaluate M_0, M_1, and N in parallel for the evaluation of $M = \langle\!\langle 0x \to M_0; 1x \to M_1 \rangle\!\rangle\ N$, the procedure we mentioned above is almost sequential in that the evaluations of M_0 and M_1 are expected to terminate because they contain the free vaiable x in many cases. There are some cases that the evaluation of M_0 does not terminate and it outputs infinitely many digits. However, if M_0 has the form d_0d_1M', then, from the forms of (OUT 1) to (OUT 4), one can consider that M_0 has enough outputs for M to make an output and terminate its reduction and proceed to M_1. We also need to take care of the case M_0 has the form $\langle\!\langle 0y \to M_{00}; 1y \to M_{11} \rangle\!\rangle\ L$. In this case, if we reduce M according to the procedure we mentioned above, and L is reduced to $\overline{0}L_1 \rhd^* \overline{0}^2 L_2 \rhd^* \cdots \rhd^* \overline{0}^n L_n$, for examle, then one repeats the application of (COND $\overline{0}$) and the outputs of N are not instantiated to x. However, we can handle many of the cases by defining that $\langle\!\langle 0y \to M_{00}; 1y \to M_{11} \rangle\!\rangle\ L$ cannot be reduced if M_{00} and M_{11} cannot be reduced and all the appearances of y in M_{00} and M_{11} have the form $\mathsf{c}_0\mathsf{c}_1\ldots\mathsf{c}_k y$ for $k > 1$ and $\mathsf{c}_i \in \{\mathsf{0},\mathsf{1}\}$. In this case, further digits of y do not change the situation that M_{00} cannot be reduced. In this way, we designed a sequential reduction strategy of XPCF. We implemented it with Haskell. As it is proved in [7], in an interval domain model, an adequate real number calculus in which average function is definable does not have a sequential reduction system. It is also the case in our model and this sequential strategy is not adequate. Therefore, it does not evaluate all the terms to their denotations. However, we observed that applications of terms in Section 4 are reduced with our implementation and we expect that it evaluates many of the "meaningful" terms to their semantics.

6 Computational adequacy of XPCF

We show the soundness and completeness properties of XPCF and we first show that two kinds of substitutions in the reduction rule of XPCF preserve meanings.

Lemma 6.1 (i) *For terms $M : \tau$ and $N : \sigma$, a variable x^σ, and an environment ρ, $[\![M[N/x^\sigma]]\!](\rho) = [\![M]\!](\rho[[\![N]\!]/x^\sigma])$.*

(ii) *For a term M and $\mathsf{b} \in \{\mathsf{0},\mathsf{1},\bar{\mathsf{0}},\bar{\mathsf{1}}\}$, $[\![M[\mathsf{b}x/x]]\!](\rho) = [\![M]\!](\rho[[\![\mathsf{b}]\!]\rho(x)/x])$.*

Proof. By structural induction on M. □

From Lemma 6.1, the following proposition holds.

Proposition 6.2 *For XPCF terms M, N and an environment ρ, if $M \rhd N$ then $[\![M]\!](\rho) = [\![N]\!](\rho)$.*

Proof. It is proved by showing that the denotational semantics of the left side and the right side coincide for every reduction rule. □

In XPCF, we consider non-terminating computations which output digits in $\{\mathsf{0},\mathsf{1},\bar{\mathsf{0}},\bar{\mathsf{1}}\}$ as $M \rhd \ldots \rhd \mathsf{d}_0(\mathsf{d}_1 \cdots (\mathsf{d}_{n-1}M')) \rhd \ldots$. Note that the sequence $\mathsf{d}_0, \mathsf{d}_1, \cdots$ is not determined uniquely by M. For example, the term $M = \langle\!\langle \mathsf{0}x \to \mathsf{0}(\mathsf{Y}\mathsf{0}); \mathsf{1}x \to \mathsf{1}(\mathsf{Y}\mathsf{0})\rangle\!\rangle(\mathsf{1}\Omega_{\mathsf{S}})$ for $\Omega_{\mathsf{S}} = \mathsf{Y}_{\mathsf{S}\to\mathsf{S}}(\lambda x^{\mathsf{S}}.x^{\mathsf{S}})$ is reduced to $\mathsf{1}\mathsf{0}^n M'$ for every n and also to $\bar{\mathsf{0}}^n N'$ for every n. However, from Proposition 6.2, if $M \rhd^* \mathsf{d}_0(\mathsf{d}_1 \cdots (\mathsf{d}_{n-1}M'))$, then we have $d_0(d_1 ... (d_{n-1}\epsilon)) \sqsubseteq d_0(d_1 ... (d_{n-1}[\![M']\!])) = [\![M]\!]$. Therefore, the outputs are bounded by the denotation $[\![M]\!]$ of M and thus compatible. We define the evaluation $\mathbf{Eval}(M)$ of an XPCF program M of type S as follows in addition to the definitions of $\mathbf{Eval}$ to programs of type B and N in Section 3.1.

Definition 6.3 (1) For a term $M : \mathsf{S}$, we define

$$\mathbf{ev}(M) = \{d_0(d_1 ... (d_{n-1}(\epsilon))) \mid M \rhd^* \mathsf{d}_0(\mathsf{d}_1 \cdots (\mathsf{d}_{n-1}M')) \text{ for some } M'\}.$$

Here, $\mathsf{d}_i \in \{\mathsf{0},\mathsf{1},\bar{\mathsf{0}},\bar{\mathsf{1}}\}$ and $d_i = [\![\mathsf{d}_i]\!]$ for $0 \le i < n$.

(2) For an XPCF program M of type S, we define $\mathbf{Eval}(M) = \bigsqcup \mathbf{ev}(M)$.

The soundness of XPCF is derived from Proposition 6.2 immediately.

Theorem 6.4 (Soundness of XPCF) *For a program M, $\mathbf{Eval}(M) \sqsubseteq [\![M]\!]$.*

To show the completeness, we use the computability method (see [11]). That is, define the set Comp_σ of computable terms of type σ for each type σ and then show that all the XPCF terms are computable.

Definition 6.5 We define the predicate Comp_σ for each type σ by induction on types.

- **(i)** Let σ be B or N. A program $M : \sigma$ has property Comp_σ if $[\![M]\!] = [\![c]\!]$ implies $\mathbf{Eval}(M) = c$.
- **(ii)** A program $M : \mathsf{S}$ has property Comp_S if $[\![M]\!] \sqsubseteq \mathbf{Eval}(M)$. That is, for any $p \in \Sigma^*_{\perp,1}$ with $p \sqsubseteq [\![M]\!]$, $p \sqsubseteq \mathbf{Eval}(M)$ holds.
- **(iii)** A closed term $M : \sigma \to \tau$ has property $\mathrm{Comp}_{\sigma\to\tau}$ if whenever $N : \sigma$ is a closed term with property Comp_σ then MN is a term with property Comp_τ.
- **(iv)** An open term $M : \sigma$ with free variables $x_1^{\sigma_1}, ..., x_n^{\sigma_n}$ has property Comp_σ if $M[N_1/x_1^{\sigma_1}]\cdots[N_n/x_n^{\sigma_n}]$ has property Comp_σ whenever $N_1, ..., N_n$ are closed terms having properties Comp_{σ_1},...,Comp_{σ_n} respectively.

We say that a term of type σ is *computable* if it has property Comp_σ.

It is immediate to show the followings. (1) If $M : \sigma \to \tau$ and $N : \sigma$ are closed computable terms, so is MN. (2) For a ground type t, a term $M : \sigma_1 \to \cdots \to \sigma_n \to \tau$ is computable if and only if $\tilde{M}N_1 \cdots N_n$ is computable for all closed computable terms $N_1 : \sigma_1, ..., N_n : \sigma_n$ and closed instantiation $\tilde{M}$ of M by computable terms.

For $s \in \Sigma^*_{\perp,1}$, we define the context $\underline{s}[X]$ as follows,

$$\underline{s}[X] = \begin{cases} \mathsf{b}_0(\mathsf{b}_1 \cdots (\mathsf{b}_{n-1}X)) & \text{if } s = b_0b_1\cdots b_{n-1}, \\ \mathsf{b}_0(\mathsf{b}_1 \cdots \mathsf{b}_{n-1}(\overline{\mathsf{c}_0}(\overline{\mathsf{c}_1} \cdots (\overline{\mathsf{c}_{m-1}}X)))) & \text{if } s = b_0b_1\cdots b_{n-1}\perp c_0c_1\cdots c_{m-1}. \end{cases}$$

Here, $\mathsf{b}_i, \mathsf{c}_j \in \{\mathsf{0}, \mathsf{1}\}$, $b_i = [\![\mathsf{b}_i]\!]$, and $c_j = [\![\mathsf{c}_j]\!]$ for $0 \le i < n$ and $0 \le j < m$. We say that a term M of type S *outputs* $s \in \Sigma^*_{\perp,1}$ if there is a reduction $M \rhd^* \underline{s}[M']$ for some M'.

Lemma 6.6 *Let $M : \mathsf{S}$ be a computable term such that x is the only free variable and let $s \in \Sigma^*$. For any $p \sqsubseteq [\![M]\!](\rho[s/x])$ with $p \in \Sigma^*_{\perp,1}$, $M[\underline{s}[x]/x]$ outputs $s' \in \Sigma^*_{\perp,1}$ such that $p \sqsubseteq s'$*

Proof. We have $[\![M]\!](\rho[\epsilon/x]) = [\![M[\Omega_\mathsf{S}/x]]\!]$ by structural induction on M. From the equation $[\![\underline{s}[\Omega_\mathsf{S}]]\!] = s$ and Lemma 6.1 (i), we have

$$[\![M]\!](\rho[s/x]) = [\![M]\!](\rho[[\![\underline{s}[\Omega_\mathsf{S}]]\!]/x]) = [\![M[\underline{s}[\Omega_\mathsf{S}]/x]]\!].$$

Since M and $\underline{s}[\Omega_\mathsf{S}]$ are computable, $M[\underline{s}[\Omega_\mathsf{S}]/x]$ is computable. Therefore, for any $p \in \Sigma^*_{\perp,1}$ with $p \sqsubseteq [\![M]\!](\rho[s/x])$, there exists a reduction $M[\underline{s}[\Omega_\mathsf{S}]/x] \rhd^* \underline{s'}[M']$ with $s' \in \Sigma^*_{\perp,1}$ such that $p \sqsubseteq s'$. If there is a reduction sequence $M[\underline{s}[\Omega_\mathsf{S}]/x] \rhd^* \underline{s'}[M']$, then there is a reduction sequence $M[\underline{s}[x]/x] \rhd^* \underline{s'}[M']$ by ignoring the reductions related to Ω_S. Therefore, $M[\underline{s}[x]/x]$ outputs s' such that $p \sqsubseteq s'$. □

Proposition 6.7 *Every XPCF term is computable.*

Proof. We prove it by structural induction on terms.

In order to prove the computability of Y_σ for an XPCF type σ, we use an extension of the syntactic information order in [11], which we omit here. We only explain the proof of the cases $\mathsf{0}, \mathsf{1}, \bar{\mathsf{0}}, \bar{\mathsf{1}}$, $\langle \mathsf{0}x \to M_0; \mathsf{1}x \to M_1 \rangle$ and $\langle\!\langle \mathsf{0}x \to M_0; \mathsf{1}x \to M_1 \rangle\!\rangle$.

The case $\mathsf{d} \in \{\mathsf{0}, \mathsf{1}, \bar{\mathsf{0}}, \bar{\mathsf{1}}\}$. We show that for any computable term M of type S, $\mathsf{d}M$ is computable. Because the function $[\![\mathsf{d}]\!] : D_\mathsf{S} \to D_\mathsf{S}$ is continuous, for any $p \in \Sigma^*_{\perp,1}$ with $p \sqsubseteq [\![\mathsf{d}M]\!] = [\![\mathsf{d}]\!]([\![M]\!])$, there exists $q \in \Sigma^*_{\perp,1}$ with $q \sqsubseteq [\![M]\!]$ such that $p \sqsubseteq [\![\mathsf{d}]\!](q)$. Since M is computable, M outputs $s \in \Sigma^*_{\perp,1}$ such that $q \sqsubseteq s$. Therefore, $\mathsf{d}M$ outputs $[\![\mathsf{d}]\!](s)$ which satisfies $p \sqsubseteq [\![\mathsf{d}]\!](q) \sqsubseteq [\![\mathsf{d}]\!](s)$ and thus d is computable.

We show that if terms M_0 and M_1 of type σ are computable, so is the term $\langle \mathsf{0}x \to M_0; \mathsf{1}x \to M_1 \rangle$. It is enough to show that the term $\langle \mathsf{0}x \to \tilde{M}_0; \mathsf{1}x \to \tilde{M}_1 \rangle N_1 N_2 \cdots N_n$ of type a ground type τ is computable when $N_1 : \mathsf{S}, N_2, ..., N_n$ are closed computable terms and $\tilde{M}_0$ and $\tilde{M}_1$ are instantiations of all free variables, except x, of M_0 and M_1 by closed computable terms, respectively. We only show the case $\tau = \mathsf{S}$.

Case $[\![N_1]\!] = \epsilon$. From the reduction rule, we have the following equation:

$$\begin{aligned}
&[\![\langle \mathsf{0}x \to \tilde{M}_0; \mathsf{1}x \to \tilde{M}_1 \rangle N_1 N_2 \cdots N_n]\!] \\
&\qquad = [\![\langle \mathsf{0}x \to \tilde{M}_0; \mathsf{1}x \to \tilde{M}_1 \rangle]\!](\epsilon)([\![N_2]\!]) \cdots ([\![N_n]\!]) \\
&\qquad = \perp_\sigma([\![N_2]\!]) \cdots ([\![N_n]\!]) = \perp_\mathsf{S}.
\end{aligned}$$

Therefore, $\langle \mathsf{0}x \to \tilde{M}_0; \mathsf{1}x \to \tilde{M}_1 \rangle N_1 N_2 \cdots N_n$ is computable.

Case $[\![N_1]\!] = 0s$. For any $p \in \Sigma^*_{\perp,1}$ such that $p \sqsubseteq [\![\langle \mathsf{0}x \to \tilde{M}_0; \mathsf{1}x \to \tilde{M}_1 \rangle N_1 \cdots N_n]\!] = [\![\tilde{M}_0]\!]([\![N_2]\!]) \cdots ([\![N_n]\!])\rho([s/x])$, from the continuity, there exists $s' \in \Sigma^*_{\perp,1}$ such that $p \sqsubseteq [\![\tilde{M}_0([\![N_2]\!]) \cdots ([\![N_n]\!])\rho([s'/x])]\!]$ and $0s' \sqsubseteq [\![N_1]\!]$. From the computability of N_1, there exists $0s'' \in \Sigma^*_{\perp,1}$ such that N_1 outputs $0s''$ and $0s' \sqsubseteq 0s''$. Then, $p \sqsubseteq [\![\tilde{M}_0([\![N_2]\!]) \cdots ([\![N_n]\!])]\!]\rho([s''/x])$ holds. Since we have $[\![\tilde{M}_0[\underline{s''}[\Omega_\mathsf{S}]/x]N_2 \cdots N_n]\!] = [\![\tilde{M}_0]\!]([\![N_2]\!]) \cdots ([\![N_n]\!])\rho([s''/x])$ and $\underline{s''}[\Omega_\mathsf{S}]$ is computable, $\tilde{M}_0[\underline{s''}[\Omega_\mathsf{S}]/x]N_2 \cdots N_n$ is also computable and outputs $t \in \Sigma^*_{\perp,1}$ such that $p \sqsubseteq t$. Therefore, we have a reduction sequence $\langle \mathsf{0}x \to \tilde{M}_0; \mathsf{1}x \to \tilde{M}_1 \rangle N_1 \cdots N_n \rhd^* \langle \mathsf{0}x \to \tilde{M}_0; \mathsf{1}x \to \tilde{M}_1 \rangle \underline{0s''}[N_1'] \cdots N_n \rhd^* \underline{t}[M'']$ such that $p \sqsubseteq t$.

Case $[\![N_1]\!] = 1s$. The proof is similar to the case $[\![N_1]\!] = 0s$.

Case $[\![N_1]\!] = \perp u$ with $u \in \Sigma^\infty \backslash \{\epsilon\}$. From the reduction rule, we have the following equation:

$$\begin{aligned}
&[\![\langle \mathsf{0}x \to \tilde{M}_0; \mathsf{1}x \to \tilde{M}_1 \rangle N_1 \cdots N_n]\!] \\
&\qquad = [\![\langle \mathsf{0}x \to \tilde{M}_0 N_2 \cdots N_n; \mathsf{1}x \to \tilde{M}_1 N_2 \cdots N_n \rangle N_1]\!] \\
&\qquad = [\![\langle \mathsf{0}x \to \tilde{M}_0 N_2 \cdots N_n; \mathsf{1}x \to \tilde{M}_1 N_2 \cdots N_n \rangle]\!](\perp u) \\
&\qquad = [\![\tilde{M}_0 N_2 \cdots N_n]\!](\rho[u/x]) \sqcap [\![\tilde{M}_1 N_2 \cdots N_n]\!](\rho[u/x]).
\end{aligned}$$

Because of the continuity of $[\![\tilde{M}_0 N_2 \cdots N_n]\!]$ and $[\![\tilde{M}_1 N_2 \cdots N_n]\!]$, for any $p \in \Sigma^*_{\perp,1}$ with $p \sqsubseteq [\![\langle \mathsf{0}x \to \tilde{M}_0; \mathsf{1}x \to \tilde{M}_1 \rangle N_1 \cdots N_n]\!]$, there is $s \in \Sigma^*$ such that $s \sqsubseteq u$, $p \sqsubseteq [\![\tilde{M}_0 N_2 \cdots N_n]\!](\rho[s/x])$, and $p \sqsubseteq [\![\tilde{M}_1 N_2 \cdots N_n]\!](\rho[s/x])$. Since N_1 is computable, N_1 outputs $\perp s$. By Lemma 6.6, $(\tilde{M}_0 N_2 \cdots N_n)[\underline{s}[x]/x]$ outputs $q_0 \in \Sigma^*_{\perp,1}$ such that $p \sqsubseteq q_0$ and $(\tilde{M}_1 N_2 \cdots N_n)[\underline{s}[x]/x]$ outputs $q_1 \in \Sigma^*_{\perp,1}$ such that $p \sqsubseteq q_1$. Therefore,

$\langle \mathsf{0}x \to \tilde{M}_0; \mathsf{1}x \to \tilde{M}_1\rangle N_1 \cdots N_n$ has the following reduction:

$$\begin{aligned}
&\langle \mathsf{0}x \to \tilde{M}_0; \mathsf{1}x \to \tilde{M}_1\rangle N_1 \cdots N_n\\
&\quad \rhd^* \langle \mathsf{0}x \to \tilde{M}_0 N_2 \cdots N_n; \mathsf{1}x \to \tilde{M}_1 N_2 \cdots N_n\rangle \underline{\bot s}[N_1']\\
&\quad \rhd^* \langle\!\langle \mathsf{0}x \to (\tilde{M}_0 N_2 \cdots N_n)[\underline{s}[x]/x]; \mathsf{1}x \to (\tilde{M}_1 N_2 \cdots N_n)[\underline{s}[x]/x]\rangle\!\rangle N_1'\\
&\quad \rhd^* \langle\!\langle \mathsf{0}x \to \underline{q_0}[M_0']; \mathsf{1}x \to \underline{q_1}[M_1']\rangle\!\rangle N_1'\\
&\quad \rhd^* \underline{q}[M']
\end{aligned}$$

for some $q \in \Sigma^*_{\bot,1}$ such that $p \sqsubseteq q \sqsubseteq (q_0 \sqcap q_1)$.

The computability proof of $\langle\!\langle \mathsf{0}x \to \tilde{M}_0; \mathsf{1}x \to \tilde{M}_1\rangle\!\rangle$ is that of $\langle \mathsf{0}x \to M_0; \mathsf{1}x \to M_1\rangle$ without the case $[\![N_1]\!] = \epsilon$ and without ristricting in the final case to $u \not\in \epsilon$. □

Therefore, the completeness of XPCF holds.

Theorem 6.8 (Completeness of XPCF) *For a program M,* $\mathbf{Eval}(M) \sqsupseteq [\![M]\!]$.

Combining the soundness and completeness of XPCF, we have the computational adequacy of XPCF. That is, $\mathbf{Eval}(M) = [\![M]\!]$ for every program M.

7 Expressive power of XPCF

In this section, we often omit the type and write x for x^σ when no confusion can arise.

We compare expressive powers of XPCF and PCF$^+$. Here, PCF$^+$ is the calculus PCF extended with the parallel conditional $\mathsf{pif}_\sigma : \mathsf{B} \to \sigma \to \sigma \to \sigma$ as a constant for each $\sigma \in \{\mathsf{B}, \mathsf{N}\}$. The interpretation of pif_σ is given as follows

$$[\![\mathsf{pif}_\sigma]\!] = \lambda b \in D_{\mathsf{B}}.\lambda x \in D_\sigma.\lambda y \in D_\sigma.\begin{cases} x & (b = tt)\\ y & (b = ff)\\ x & (b = \bot_{\mathsf{B}} \text{ and } x = y)\\ \bot_\sigma & (\text{otherwise}).\end{cases}$$

The operational semantics of PCF$^+$ is the operational semantics of PCF together with:

$$\begin{array}{ccc}
\mathsf{pif}_\sigma\, M\, c\, c \rhd c, & \mathsf{pif}_\sigma\, \mathsf{tt}\, M\, N \rhd M & \mathsf{pif}_\sigma\, \mathsf{ff}\, M\, N \rhd N\\
\dfrac{M \rhd M'}{\mathsf{pif}_\sigma\, M \rhd \mathsf{pif}_\sigma\, M'} & \dfrac{N \rhd N'}{\mathsf{pif}_\sigma\, M\, N \rhd \mathsf{pif}_\sigma\, M\, N'} & \dfrac{L \rhd L'}{\mathsf{pif}_\sigma\, M\, N\, L \rhd \mathsf{pif}_\sigma\, M\, N\, L'}.
\end{array}$$

Consider the following XPCF term pif'_σ of type $\mathsf{B} \to \sigma \to \sigma \to \sigma$ for $\sigma \in \{\mathsf{B}, \mathsf{N}\}$.

$$\mathsf{pif}'_\sigma = \lambda u^{\mathsf{B}}.\lambda y^\sigma.\lambda z^\sigma.\langle\!\langle \mathsf{0}x \to y; \mathsf{1}x \to z\rangle\!\rangle(\mathsf{if}_{\mathsf{S}}\, u\, (\mathsf{0}\Omega_{\mathsf{S}})\, (\mathsf{1}\Omega_{\mathsf{S}})).$$

It satisfies $[\![\mathsf{pif}']\!] = [\![\mathsf{pif}]\!]$ and therefore it expresses the pif_σ operator. Note that one can also express it as as XPCF program

$$\lambda u^{\mathsf{B}}.\lambda y^\sigma.\lambda z^\sigma.\langle \mathsf{0}x \to y; \mathsf{1}x \to z\rangle \overline{\mathsf{0}}(\mathsf{if}_{\mathsf{S}}\, u\, (\mathsf{0}\Omega_{\mathsf{S}})\, (\mathsf{1}\Omega_{\mathsf{S}}))$$

without using $\langle\!\langle ... \rangle\!\rangle$. Thus, PCF$^+$ terms can be translated into XPCF terms by replacing pif_σ with pif'_σ.

Theorem 7.1 *For a PCF$^+$ term $M : \sigma$ and a PCF environment ρ, there exists an XPCF term $M' : \sigma$ such that $[\![M]\!](\rho) = [\![M']\!](\rho')$. Here, ρ' is any extension of ρ to an XPCF environment.*

On the other hand, there is an embedding-projection pair (e-p pair in short) (e, p) between the domains $D_{\mathsf{S}} = \mathbf{BD}$ and $D_{\mathsf{N}\to\mathsf{B}} \cong \Sigma_\perp^\omega$ where the projection p is the function trunc in Section 2.2. Here, a pair of continuous functions $e : X \to Y$ and $p : Y \to X$ is an e-p pair if they satisfy $p \circ e = id_X$ and $e \circ p \sqsubseteq id_Y$. Terms $\mathbf{e} : \mathsf{S} \to (\mathsf{N} \to \mathsf{B})$ and $\mathbf{p} : (\mathsf{N} \to \mathsf{B}) \to \mathsf{S}$ such that $[\![\mathbf{e}]\!] = e$ and $[\![\mathbf{p}]\!] = p$ can be written in XPCF as follows,

$$\begin{aligned}
\mathbf{e} :&= \mathsf{Y}_{\mathsf{S}\to(\mathsf{N}\to\mathsf{B})}(\lambda f^{\mathsf{S}\to\mathsf{N}\to\mathsf{B}}.\lambda g^{\mathsf{S}}.\lambda n^{\mathsf{N}}.\mathsf{if}\,(\mathsf{zero}\,n)\,(\mathbf{head}\,g)\,(f\,(\mathbf{tail}\,g)\,(\mathsf{dec}\,n)))\\
\mathbf{p} :&= \mathsf{Y}_{\mathsf{N}\to(\mathsf{N}\to\mathsf{B})\to\mathsf{S}}(\lambda g^{\mathsf{N}\to(\mathsf{N}\to\mathsf{B})\to\mathsf{S}}.\lambda n^{\mathsf{N}}.\lambda f^{\mathsf{N}\to\mathsf{B}}.\langle 0x\to 0(g\,(\mathsf{inc}\,n)\,f); 1x\to 1(g\,(\mathsf{inc}\,n)\,f)\rangle\\
&\qquad\qquad\qquad \bar{0}(\mathsf{if}(f\,n)(0\Omega_{\mathsf{S}})(1\Omega_{\mathsf{S}})))\mathsf{k}_0
\end{aligned}$$

where $\mathbf{tail} = \langle 0x\to x; 1x\to x\rangle$ and $\mathbf{head} = \langle 0x\to \mathsf{tt}; 1x\to \mathsf{ff}\rangle$.

We can extend the e-p pair (e, p) to higher order types. We inductively define σ^t for every XPCF type σ as follows

$$\mathsf{B}^t = \mathsf{B}, \mathsf{N}^t = \mathsf{N}, \mathsf{S}^t = \mathsf{N} \to \mathsf{B}, \text{and } (\sigma \to \tau)^t = \sigma^t \to \tau^t.$$

We inductively define $\mathbf{e}_\sigma : \sigma \to \sigma^t$ and $\mathbf{p}_\sigma : \sigma^t \to \sigma$ for every XPCF type σ as follows,

$$\begin{aligned}
\mathbf{e}_{\mathsf{N}} = \mathbf{p}_{\mathsf{N}} &= \lambda x^{\mathsf{N}}.x, \quad \mathbf{e}_{\mathsf{B}} = \mathbf{p}_{\mathsf{B}} = \lambda x^{\mathsf{B}}.x, \quad \mathbf{e}_{\mathsf{S}} = \mathbf{e}, \quad \mathbf{p}_{\mathsf{S}} = \mathbf{p},\\
\mathbf{e}_{\sigma\to\tau} &= \lambda f^{\sigma\to\tau}.\lambda x^{\sigma^t}.\mathbf{e}_\tau(f(\mathbf{p}_\sigma(x))),\\
\mathbf{p}_{\sigma\to\tau} &= \lambda f^{\sigma^t\to\tau^t}.\lambda x^{\sigma}.\mathbf{p}_\tau(f(\mathbf{e}_\sigma(x))).
\end{aligned}$$

It is immediate to show that $([\![\mathbf{e}_\sigma]\!], [\![\mathbf{p}_\sigma]\!])$ is e-p pair for every type σ.

We define a syntactical translation $(-)^t$ from XPCF terms to PCF$^+$ terms so that $M^t : \sigma^t$ for $M : \sigma$. Before that, we define a function $r : D_{\mathsf{N}\to\mathsf{B}} \to D_{\mathsf{N}\to\mathsf{B}}$ as $r = e \circ p$ and $r_\sigma : D_{\sigma^t} \to D_{\sigma^t}$ as $r_\sigma = e_\sigma \circ p_\sigma$. The function r satisfies

$$r(f)(n) = \begin{cases} tt & \text{if } f(n) = tt \text{ and } \perp \text{ appears at most once in } f(0), \cdots, f(n-1)\\ ff & \text{if } f(n) = ff \text{ and } \perp \text{ appears at most once in } f(0), \cdots, f(n-1)\\ \perp & \text{otherwise.}\end{cases}$$

for $f : D_{\mathsf{N}} \to D_{\mathsf{B}}$ and $n \in D_{\mathsf{N}}$. Let $\mathbf{r}$ be any PCF$^+$ term such that $[\![\mathbf{r}]\!] = r$. We inductively define a PCF$^+$ term $\mathbf{r}_\sigma : \sigma^t \to \sigma^t$ which satisfies $[\![\mathbf{r}_\sigma]\!] = r_\sigma$ for every XPCF type σ as follows

$$\mathbf{r}_{\mathsf{B}} = \lambda x^{\mathsf{B}}.x^{\mathsf{B}}, \mathbf{r}_{\mathsf{N}} := \lambda x^{\mathsf{N}}.x^{\mathsf{N}}, \mathbf{r}_{\mathsf{S}} := \mathbf{r}, \text{and } \mathbf{r}_{\sigma\to\tau} = \lambda f^{\sigma^t\to\tau^t}.\lambda x^{(\sigma^t)}.\mathbf{r}_\tau(f(\mathbf{r}_\sigma x)).$$

For an XPCF term M, we inductively define $M^t : \sigma^t$ as follows,

$$\begin{aligned}
(x^\sigma)^t = \mathbf{r}_\sigma x^{(\sigma^t)}, \quad \mathsf{c}^t = \mathsf{c}, \quad \mathsf{if}^t_\sigma = \mathsf{if}_\sigma, \quad \mathsf{Y}^t_\sigma = \lambda f^{\sigma^t \to \sigma^t}.\mathsf{Y}_{\sigma^t}(\mathbf{r}_{\sigma\to\sigma} f),\\
(\lambda x^\sigma.M)^t = \lambda x^{(\sigma^t)}.M : t[\mathbf{r}_\sigma(x^{(\sigma^t)})/x^\sigma], \quad (MN)^t = (M^t N^t),\\
\mathsf{0}^t &= \lambda f^{\mathsf{N}\to\mathsf{B}}.\lambda x^{\mathsf{N}}.\mathsf{if}\,(\mathsf{zero}\,x)\,\mathsf{tt}\,((\mathbf{r}_{\mathsf{S}}\,f)\,(\mathsf{dec}\,x))\\
\mathsf{1}^t &= \lambda f^{\mathsf{N}\to\mathsf{B}}.\lambda x^{\mathsf{N}}.\mathsf{if}\,(\mathsf{zero}\,x)\,\mathsf{ff}\,((\mathbf{r}_{\mathsf{S}}\,f)\,(\mathsf{dec}\,x))\\
\overline{\mathsf{0}}^t &= \lambda f^{\mathsf{N}\to\mathsf{B}}.\lambda x^{\mathsf{N}}.\mathsf{if}\,(\mathsf{zero}\,x)\,((\mathbf{r}_{\mathsf{S}}\,f)\,\mathsf{k}_0)\\
&\qquad\qquad (\mathsf{if}\,(\mathsf{zero}\,(\mathsf{dec}\,x))\,\mathsf{tt}\,((\mathbf{r}_{\mathsf{S}}\,f)\,(\mathsf{dec}\,x)))\\
\overline{\mathsf{1}}^t &= \lambda f^{\mathsf{N}\to\mathsf{B}}.\lambda x^{\mathsf{N}}.\mathsf{if}\,(\mathsf{zero}\,x)\,((\mathbf{r}_{\mathsf{S}}\,f)\,\mathsf{k}_0)\\
&\qquad\qquad (\mathsf{if}\,(\mathsf{zero}\,(\mathsf{dec}\,x))\,\mathsf{ff}\,((\mathbf{r}_{\mathsf{S}}\,f)\,(\mathsf{dec}\,x)))\\
\langle\!\langle \mathsf{0}x \to M_0; \mathsf{1}x \to M_1 \rangle\!\rangle^t &= \lambda f^{\mathsf{N}\to\mathsf{B}}.\mathsf{pif}\,((\mathbf{r}_{\mathsf{S}}\,f)\,\mathsf{k}_0)\,M_0^t[\lambda y^{\mathsf{N}}.(\mathbf{r}_{\mathsf{S}}\,f)(\mathsf{inc}\,y)/x^{\mathsf{N}\to\mathsf{B}}]\\
&\qquad\qquad M_1^t[\lambda y^{\mathsf{N}}.(\mathbf{r}_{\mathsf{S}}\,f)(\mathsf{inc}\,y)/x^{\mathsf{N}\to\mathsf{B}}],\\
\langle \mathsf{0}x \to M_0; \mathsf{1}x \to M_1 \rangle^t &= \lambda f^{\mathsf{N}\to\mathsf{B}}.\mathsf{if}(\mathsf{pif}\,(\mathsf{if}\,((\mathbf{r}_{\mathsf{S}}\,f)\,\mathsf{k}_0)\,\mathsf{tt}\,\mathsf{tt})\,\mathsf{tt}\,(\mathsf{if}\,((\mathbf{r}_{\mathsf{S}}\,f)\,\mathsf{k}_1)\,\mathsf{tt}\,\mathsf{tt}))\\
&\qquad (\mathsf{pif}\,((\mathbf{r}_{\mathsf{S}}\,f)\,\mathsf{k}_0)\,M_0^t[\lambda y^{\mathsf{N}}.(\mathbf{r}_{\mathsf{S}}\,f)(\mathsf{inc}\,y)/x^{\mathsf{N}\to\mathsf{B}}]\\
&\qquad\qquad M_1^t[\lambda y^{\mathsf{N}}.(\mathbf{r}_{\mathsf{S}}\,f)(\mathsf{inc}\,y)/x^{\mathsf{N}\to\mathsf{B}}])\,\Omega_\sigma
\end{aligned}$$

where c is a constant other than $\mathsf{0}, \mathsf{1}, \overline{\mathsf{0}}, \overline{\mathsf{1}}, \mathsf{if}_\sigma$ or Y_σ and the types of terms M_0 and M_1 are both σ. Here, we assume that the same XPCF variable does not appear in different types to prevent conflictions in the translation to PCF$^+$ terms.

We define a translation $(\text{-})^t$ of environments as $\rho^t(x^{\sigma^t}) = e_\sigma(\rho(x^\sigma))$.

Proposition 7.2 *For any term $M : \sigma$ in XPCF and environment ρ, $e_\sigma([\![M]\!](\rho)) = [\![M^t]\!](\rho^t)$ holds.*

Proof. By structual induction on M. □

Theorem 7.3 (i) *XPCF and PCF$^+$ have the same expressive power on PCF types.*

(ii) *All computable elements of D_{S} are definable in XPCF.*

(iii) *The function exist is not definable in XPCF. Here, exist is the function $D_{\mathsf{N}\to\mathsf{B}} \to D_{\mathsf{B}}$ which satisfies*

$$exist = \lambda f \in D_{\mathsf{N}\to\mathsf{B}}.\begin{cases} ff & f(\bot) = ff \\ tt & \exists n \in \mathbb{N}. f(n) = tt \\ \bot & otherwise. \end{cases}$$

Proof. (i) Since e_σ is the identity function if σ does not contain S, Theorem 7.1 and Proposition 7.2 show that XPCF and PCF$^+$ have the same expressive power on PCF types.

(ii) For any computable element $x \in D_{\mathsf{S}}$, $e_{\mathsf{S}}(x) \in D_{\mathsf{N}\to\mathsf{B}}$ is a computable element because e_{S} is a computable function. Therefore, $e_{\mathsf{S}}(x)$ is definable in PCF$^+$ [11]. Since p_{S} is definable in XPCF, $p_{\mathsf{S}}(e_{\mathsf{S}}(x)) = x$ is definable in XPCF.

(iii) Suppose that there exists a closed XPCF term $M : (\mathsf{N} \to \mathsf{B}) \to \mathsf{B}$ such that $[\![M]\!] = exist$. By the translation, we have $e_{(\mathsf{N}\to\mathsf{B})\to\mathsf{B}}([\![M]\!]) = [\![M^t]\!]$. However, *exist* is not definable in PCF $^+$ [11] and this is a contradiction. Therefore, exist is not definable in XPCF. □

In [11], Plotkin introduced the language PCF $^{++}$ which is an extension of PCF $^+$ by adding the existential quantifier $\exists : (\mathsf{N} \to \mathsf{B}) \to \mathsf{B}$ as a constant such that $[\![\exists]\!] = exist$. [5] showed that Real PCF extended with $\exists$ is universal, based on a technique due to Thomas Streicher [13] to establish that PCF extended with recursive types, parallel-or and $\exists$ is universal. We define a calculus XPCF $^\exists$, which is the extension of XPCF with the $\exists$ operator. XPCF $^\exists$ is universal in the following sense.

Theorem 7.4 *For every XPCF type σ, all computable elements of D_σ are definable in XPCF$^\exists$.*

Proof. For any XPCF type σ and computable element $x \in D_\sigma$, $e_\sigma(x) \in D_{\sigma^t}$ is a computable element because e_σ is a computable function. Therefore, $e_\sigma(x)$ is definable in PCF $^{++}$ by [11]. Since p_σ is definable in XPCF, $p_\sigma(e_\sigma(x)) = x$ is definable in XPCF $^\exists$. □

Acknowledgements. The authors would like to thank to Takashi Sakuragawa for valuable comments. This work was partly supported by JSPS KAKENHI Grant Number 22500014.

References

[1] Berger, U., and Hou, T., *Coinduction for Exact Real Number Computation.* Theory of Computing Systems. **43** (2008), 394-409.

[2] Boehm, H.J., Cartwright, R., Riggle, M., and O'Donnell, M.J., "Exact real arithmetic: a case study in higher order programming", ACM Symposium on Lisp and Functional Programming, ACM, New York, 1986.

[3] Blanck, J., *Domain representations of topological spaces.* Theoritical Computer Science. **247** (2000), 229–255.

[4] Brattka, V., and Hertling, P., *Topological properties of real number representations.* Theoretical Computer Science. **284** (2002), 241-257.

[5] Escardó, M.H., *Real PCF extended with ∃ is universal.* Advances in Theory and Formal Methods of Computing:Proceedings of the Third Imperial Collage Workshop. (1996), 13-24.

[6] Escardó, M.H., *PCF extended with real numbers.* Theoretical Computer Science. **162** (1996), 79-115.

[7] Escardó, M.H., Hofmann, M., and Streicher, T., *On the non-sequential nature of the interval-domain model of real-number computation*, Math. Struct. in Comp. Science. **14**(6) (2004), 803-814.

[8] Di Gianantonio, P., *An abstract data type for real numbers.* Theoretical Computer Science. **221** (1999), 295-326.

[9] Gierz, G., Hofmann, K.H., Keimel, K., Lawson, J.D., Mislove, M., and Scott, D.S., "Continuous Lattices and Domains", Cambridge University Press, 2003.

[10] Gray, F., "Pulse code communication", March 17, 1953 (filed Nov. 1947). U.S. Patent 2,632,058.

[11] Plotkin, G., *LCF considered as a programming language*, Theoretical Computer Science. **5** (1977), 223-255.

[12] Schwichtenberg, H., and Wainer, S.S., *Proofs and Computations*, Cambridge University Press, 2012.

[13] Streicher, T., *A universality theorem for PCF with recursive types, parallel-or and ∃.* Mathematical Structures for Computing Science. **4**(1) (1994), 111-115.

[14] Streicher, T., “Domain-Theoretic Foundations of Functional Programming”, World Scientific, 2006.

[15] Tsuiki, H., *Real number computation through gray code embedding*. Theoretical Computer Science. **284**(2) (2002), 467-485.

[16] Tsuiki, H., *Compact metric spaces as minimal-limit sets in domains of bottomed sequences*. Math. Struct. in Comp. Science. **14** (2004), 853-878.

[17] Weihrauch, K., “An Introduction to Computable Analysis”, Springer, Berlin, 2000.

Distributed Probabilistic Strategies

Glynn Winskel [1]

University of Cambridge Computer Laboratory, England

Abstract

Building on a new definition and characterization of probabilistic event structures, a general definition of distributed probabilistic strategies is proposed. Probabilistic strategies are shown to compose, with probabilistic copy-cat strategies as identities. A higher-order probabilistic process language reminiscent of Milner's CCS is interpreted within probabilistic strategies. W.r.t. a new definition of quantum event structure, it is shown how consistent parts of a quantum event structure are automatically probabilistic event structures, and so possess a probability measure. This gives a non-traditional take on the consistent-histories approach to quantum theory. It leads to an extension to quantum strategies. Probabilistic games extend to games with payoff, symmetry and games of imperfect information.

1 Introduction

Concurrent strategies [12] are being investigated as a possible foundation for a generalized domain theory, in which concurrent games and strategies take over the roles of domains and continuous functions. One motivation is to broaden the range of applicability of denotational semantics. Hence it is important to see how concurrent strategies can be adapted to quantitative semantics, to probabilistic and quantum strategies.

Just as event structures can be thought of as models of distributed computation so are probabilistic event structures models of probabilistic distributed processes. Existing definitions of probabilistic event structures [1,8,13] are not general enough to ascribe probabilities to the results of the sometimes partial interaction between strategies. This paper first provides a new workable definition of probabilistic event structures, extending existing definitions. Probabilistic event structures are characterized as event structures with a continuous valuation on their domain of configurations. Probabilistic event structures possess a probabilistic measure on their configurations. Technically, probabilistic event structures are defined via 'drop functions' expressing the probability drops across general intervals of configurations of the event structure; 'drop functions' provide a useful mathematical handle on probabilistic event structures and strategies.

[1] Email: Glynn.Winskel@cl.cam.ac.uk

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.com/locate/entcs

This prepares the ground for a general definition of distributed probabilistic strategies, based on event structures. A probabilistic strategy for Player is a concurrent strategy whose behaviour is described by a probabilistic event structure when projected to just the Player moves. Probabilistic strategies are shown to compose—here 'drop functions' come into their own—with probabilistic copy-cat strategies as identities. The result of a play between Player and Opponent in a game will be a probabilistic event structure.

As an illustration of their expressive power, probabilistic strategies are shown to interpret a higher-order probabilistic process language reminiscent of Milner's CCS. Probabilistic strategies are easily extended to games with payoff and games of imperfect information. Their definition has been partly inspired by the work of Danos and Harmer on probabilistic HO games [3], and in an informal sense the definition here extends theirs from the sequential setting. (A formal connection must await the relation between concurrent games and HO games, being developed within concurrent games with symmetry [2].)

A novel application is to a new definition of quantum event structures and strategies. A quantum event structure is an event structure in which the events are interpreted as projection or unitary operators on a Hilbert space, so that concurrent events are associated with commuting operators; a configuration of the event structure is thought of as a partial-order history of the observations of a quantum experiment. Interestingly order-compatible families of configurations of a quantum event structure automatically determine a probabilistic event structures, and so possess probability distributions. This gives a non-traditional take on the consistent-histories approach to quantum theory, which provides consistency conditions on histories to pick out those subfamilies of histories over which it is meaningful to place a probability distribution. The approach via quantum event structures bypasses the consistency conditions usually invoked [6]—those conditions appear to be too sensitive to what one considers the initial and final events of a finite history.

In a quantum game Player and Opponent interact to jointly create a probabilistic distributed experiment on a quantum system. Accordingly a quantum strategy is taken to be a distributed probabilistic strategy on a quantum event structure, according with work on quantum games [5]. There are similarities with the work of Delbecque [4], itself based on probabilistic HO games [3]. Full proofs can be found in [16].

2 Event structures

2.1 Event structures and configurations

An *event structure* comprises $(E, \leq, \mathrm{Con})$, consisting of a set E, of *events* which are partially ordered by $\leq$, the *causal dependency relation*, and a nonempty *consistency relation* Con consisting of finite subsets of E, which satisfy

$$\begin{aligned}
&\{e' \mid e' \leq e\} \text{ is finite for all } e \in E,\\
&\{e\} \in \mathrm{Con} \text{ for all } e \in E,\\
&Y \subseteq X \in \mathrm{Con} \implies Y \in \mathrm{Con}, \quad \text{and}\\
&X \in \mathrm{Con} \;\&\; e \leq e' \in X \implies X \cup \{e\} \in \mathrm{Con}.
\end{aligned}$$

The *configurations*, $\mathcal{C}^\infty(E)$, of an event structure E consist of those subsets $x \subseteq E$ which are *(Consistent)* $\forall X \subseteq x.\ X$ is finite $\Rightarrow X \in \mathrm{Con}$ and *(Down-closed)* $\forall e, e'.\ e' \leq e \in x \implies e' \in x$. Often we shall be concerned with just the finite configurations, $\mathcal{C}(E)$.

We say an event structure is *elementary* when the consistency relation consists of all finite subsets of events. Two events e, e' which are both consistent and incomparable w.r.t. causal dependency in an event structure are regarded as *concurrent*, written $e\ co\ e'$. In games the relation of *immediate* dependency $e \rightarrowtriangle e'$, meaning e and e' are distinct with $e \leq e'$ and no event in between, will play an important role. For $X \subseteq E$ we write $[X]$ for $\{e \in E \mid \exists e' \in X.\ e \leq e'\}$, the down-closure of X; note if $X \in \mathrm{Con}$, then $[X] \in \mathrm{Con}$ is a configuration.

Notation 1 Let E be an event structure. We use $x \mathrel{-\!\!\!\subset} y$ to mean y covers x in $\mathcal{C}^\infty(E)$, *i.e.* $x \subsetneq y$ in $\mathcal{C}^\infty(E)$ with nothing in between, and $x \overset{e}{\mathrel{-\!\!\!\subset}} y$ to mean $x \cup \{e\} = y$ for $x, y \in \mathcal{C}^\infty(E)$ and event $e \notin x$. We use $x \overset{e}{\mathrel{-\!\!\!\subset}}$, expressing that event e is enabled at configuration x, when $x \overset{e}{\mathrel{-\!\!\!\subset}} y$ for some y.

2.2 Maps and operations on event structures

Let E and E' be event structures. A *map* of event structures $f : E \to E'$ is a partial function on events $f : E \rightharpoonup E'$ such that for all $x \in \mathcal{C}^\infty(E)$ its direct image $fx \in \mathcal{C}^\infty(E')$ and

$$e_1, e_2 \in x \;\&\; f(e_1) = f(e_2) \text{ (with both defined)} \implies e_1 = e_2.$$

Maps of event structures compose as partial functions, with identity maps given by identity functions. We will say the map is *total* if the function f is total. A total map of event structures which preserves causal dependency is called *rigid.*

2.2.1 Products

The category of event structures with maps has *products* $A \times B$ with projections π_1 to A and π_2 to B. It introduces arbitrary synchronisations between events of A and events of B in the manner of process algebra.

2.2.2 Pullbacks

Synchronized compositions of event structures A and B are obtained as restrictions $A \times B \upharpoonright R$. The *restriction* of an event structure E to a subset of events R, written $E \upharpoonright R$, is the event structure with events $E' = \{e \in E \mid [e] \subseteq R\}$ and causal dependency and consistency induced by E. We obtain *pullbacks* as a special case. Let $f : A \to C$ and $g : B \to C$ be maps of event structures. Defining $P =_{\mathrm{def}} A \times B \upharpoonright \{p \in A \times B \mid f\pi_1(p) = g\pi_2(p)\}$ we obtain a pullback P, π_1, π_2 in the category of event structures. When f and g are total the same construction gives the pullback in the category of event structures with *total* maps.

2.2.3 Projection

Let $(E, \leq, \mathrm{Con})$ be an event structure. Let $V \subseteq E$ be a subset of 'visible' events. Define the *projection* of E on V, to be $E{\downarrow}V =_{\mathrm{def}} (V, \leq_V, \mathrm{Con}_V)$, where $v \leq_V v'$ iff $v \leq v'$ & $v, v' \in V$ and $X \in \mathrm{Con}_V$ iff $X \in \mathrm{Con}$ & $X \subseteq V$. A partial map $f : E \to E'$ of event structures factors into a composition of a partial and total map $E \to E{\downarrow}V \to E'$ where: $V =_{\mathrm{def}} \{e \in E \mid f(e) \text{ is defined}\}$ is the domain of definition of f; the partial map $E \to E{\downarrow}V$ acts as identity on V as is undefined otherwise; and the total map $E{\downarrow}V \to E'$ acts as f.

2.2.4 Prefixes and sums

The category of event structures has coproducts given as sums; a coproduct $\sum_{i\in I} E_i$ is obtained as the disjoint juxtaposition of an indexed collection of event structures, making events in distinct components inconsistent. In practice, components of a sum are often prefixed by an event. The prefix of an event structure A, written $\bullet.A$, comprises the event structure in which all the events of A are made to causally depend on an event $\bullet$.

3 Probabilistic event structures

A probabilistic event structure comprises an event structure $(E, \leq, \mathrm{Con})$ with a continuous valuation on its Scott open sets of configurations.[2] Continuous valuations play the role of elements in probabilistic powerdomains [7]. Continuous valuations are determined by their restrictions to basic open sets $\widehat{x} =_{\mathrm{def}} \{y \in \mathcal{C}^\infty(E) \mid x \subseteq y\}$, for x a finite configuration. This leads to an equivalent, more workable definition that we develop now. The description of a probabilistic event structure here extends the definitions mentioned in [13].

3.1 General intervals and drop functions

Throughout this section assume E is an event structure and $v : \mathcal{C}(E) \to \mathbb{R}$. Extend $\mathcal{C}(E)$ to a lattice $\mathcal{C}(E)^\top$ by adjoining an extra top element $\top$. Write its order as $x \sqsubseteq y$ and its finite join operations as $x \vee y$ and $\bigvee_{i\in I} x_i$. Extend v to $v^\top : \mathcal{C}(E)^\top \to \mathbb{R}$ by taking $v^\top(\top) = 0$.

We are concerned with drops in value across general intervals $[y; x_1, \cdots, x_n]$, where $y, x_1, \cdots, x_n \in \mathcal{C}(E)^\top$ with $y \sqsubseteq x_1, \cdots, x_n$ in $\mathcal{C}(E)^\top$. The interval is thought of as specifying the set of configurations $\widehat{y} \setminus (\widehat{x}_1 \cup \cdots \cup \widehat{x}_n)$. As such the intervals form a basis of the Lawson topology on $\mathcal{C}^\infty(E)^\top$.

[2] *Viz.* a function w from the Scott-open subsets of $\mathcal{C}^\infty(E)$ to $[0,1]$ which is *(normalized)* $w(\mathcal{C}^\infty(E)) = 1$; *(strict)* $w(\emptyset) = 0$; *(monotone)* $U \subseteq V \implies w(U) \leq w(V)$; *(modular)* $w(U \cup V) + w(U \cap V) = w(U) + w(V)$; and *(continuous)* $w(\bigcup_{i\in I} U_i) = \sup_{i\in I} w(U_i)$, for *directed* unions. The idea: $w(U)$ is the probability of a result in open set U.

Define the *drop functions* $d_v^{(n)}[y; x_1, \cdots, x_n] \in \mathbb{R}$ for $y, x_1, \cdots, x_n \in \mathcal{C}(E)^\top$ with $y \sqsubseteq x_1, \cdots, x_n$ in $\mathcal{C}(E)^\top$, by taking $d_v^{(0)}[y;] =_{\text{def}} v^\top(y)$ and

$$d_v^{(n)}[y; x_1, \cdots, x_n] =_{\text{def}} d_v^{(n-1)}[y; x_1, \cdots, x_{n-1}] - d_v^{(n-1)}[x_n; x_1 \vee x_n, \cdots, x_{n-1} \vee x_n] \,.$$

Proposition 3.1 *Let $n \in \omega$. For $y, x_1, \cdots, x_n \in \mathcal{C}(E)^\top$ with $y \sqsubseteq x_1, \cdots, x_n$,*

$$d_v^{(n)}[y; x_1, \cdots, x_n] = v(y) - \sum_{\emptyset \neq I \subseteq \{1, \cdots, n\}} (-1)^{|I|+1} v(\bigvee_{i \in I} x_i) \,.$$

It will be important that drops across general intervals can be reduced to sums of drops across intervals based on coverings, as explained next.

Lemma 3.2 *Let $y, x_1, \cdots, x_n, x'_n \in \mathcal{C}(E)^\top$ with $y \sqsubseteq x_1, \cdots, x_n$ and $x_n \sqsubseteq x'_n$, Then, $d_v^{(n)}[y; x_1, \cdots, x'_n] = d_v^{(n)}[y; x_1, \cdots, x_n] + d_v^{(n)}[x_n; x_1 \vee x_n, \cdots, x_{n-1} \vee x_n, x'_n]$.*

Corollary 3.3 *Let $y \subseteq x_1, \cdots, x_n$ in $\mathcal{C}(E)$. Then, $d_v^{(n)}[y; x_1, \cdots, x_n]$ is expressible as a sum of terms $d_v^{(k)}[u; w_1, \cdots, w_k]$ where $y \subseteq u \mathrel{-\!\!\subset} w_i$ in $\mathcal{C}(E)$ and $w_i \subseteq x_1 \cup \cdots \cup x_n$, for all i with $1 \leq i \leq k$. ($x_1 \cup \cdots \cup x_n$ need not be in $\mathcal{C}(E)$.)*

3.2 Probabilistic event structures

A probabilistic event structure is an event structure associated with a $[0, 1]$-valuation on configurations such that no general interval has a negative drop.

Definition 3.4 Let E be an event structure. A *configuration-valuation* on E is function $v : \mathcal{C}(E) \to [0, 1]$ such that $v(\emptyset) = 1$ and which satisfies the "drop condition"

$$d_v^{(n)}[y; x_1, \cdots, x_n] \geq 0$$

for all $n \geq 1$ and $y, x_1, \cdots, x_n \in \mathcal{C}(E)$ with $y \subseteq x_1, \cdots, x_n$. A *probabilistic event structure* comprises an event structure E together with a configuration-valuation $v : \mathcal{C}(E) \to [0, 1]$.

By Corollary 3.3, in showing we have a probabilistic event structure it suffices to verify the "drop condition" only for covering intervals.

Theorem 3.5 *A configuration-valuation v on an event structure E extends uniquely to a continuous valuation w_v on the open sets of $\mathcal{C}^\infty(E)$ (so $v(x) = w_v(\widehat{x})$, for all $x \in \mathcal{C}(E)$). Conversely, a continuous valuation on the open sets of $\mathcal{C}^\infty(E)$ restricts to a configuration-valuation on E.*

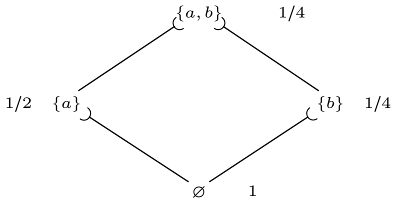
The above theorem holds (with the same proof) in greater generality, for Scott domains. Now, by [9], Corollary 4.3:

Theorem 3.6 *For a configuration-valuation v on E there is a unique probability measure μ_v on the Borel subsets of $\mathcal{C}^\infty(E)$ extending w_v.*

Example 3.7 In general, for v a configuration-valuation on E,

$$\mu_v(\{y\}) = \inf\{d_v^{(n)}[y; x_1, \cdots, x_n] \mid n \in \omega \ \& \ y \subsetneq x_1, \cdots, x_n \in \mathcal{C}(E)\}$$

for any $y \in \mathcal{C}(E)$. When $v(y) > 0$ and $\mu_v(\{y\}) = 0$ we can understand y as being a *transient* configuration on the way to a final result. In particular, consider the event structure comprising two concurrent events a and b. It has configurations and configuration valuation v as shown:



The probability $\mu_v(\{\{a,b\}\})$ of ending at the configuration $\{a,b\}$ is 1/4; that of terminating at $\{a\}$ the drop $1/2 - 1/4 = 1/4$; that of terminating at $\{b\}$ the drop $1/4 - 1/4 = 0$ showing that $\{b\}$ is only a transient configuration; while the probability of terminating at $\varnothing$ is the drop $1 - 1/2 - 1/4 + 1/4 = 1/2$. □

Remark. In the definition of probabilistic event structures there are two different ways to say, for example, that events e_1 and e_2 do not occur together at a finite configuration y where $y \overset{e_1}{-\!\!\!\subset} x_1$ and $y \overset{e_2}{-\!\!\!\subset} x_2$: either through $\{e_1, e_2\} \notin \mathrm{Con}$; or via the configuration-valuation v through $v(x_1 \cup x_2) = 0$. However, this seeming redundancy is exploited later in probabilistic strategies and quantum event structures, when we mix probability with nondeterminism and shall make use of both consistency and the valuation.

4 Probabilistic strategies

We show how concurrent strategies can be extended with probabilities, first reviewing the needed results from [12].

4.1 Strategies

4.1.1 Event structures with polarity

Both games and strategies in a game are represented in terms of event structures with polarity, which comprise (E, pol) where E is an event structure with a polarity function $pol : E \to \{+, -\}$ ascribing a polarity + (Player) or − (Opponent) to its events. The events correspond to (occurrences of) moves. Maps of event structures with polarity are maps of event structures which preserve polarities.

The *dual*, $E^\perp$, of an event structure with polarity E comprises the same underlying event structure E but with a reversal of polarities. Let A and B be event structures with polarity. The operation $A \| B$, of *simple parallel composition*, juxtaposes disjoint copies of A and B, maintaining their causal dependency and specifying a finite subset of events as consistent if it restricts to consistent subsets of A and B. Polarities are unchanged. The empty game $\varnothing$ is the unit of $\|$.

4.1.2 Pre-strategies

Let A be an event structure with polarity, thought of as a game; its events stand for the possible occurrences of moves of Player and Opponent and its causal dependency and consistency relations the constraints imposed by the game. A *pre-strategy in* A represents a nondeterministic play of the game and is defined to be a total map $\sigma : S \to A$ of event structures with polarity.

A map between pre-strategies, from $\sigma : S \to A$ and $\tau : T \to A$, is a map $f : S \to T$ such that $\sigma = \tau f$. Accordingly, $\sigma \cong \tau$ when there is an isomorphism $\theta : S \cong T$ such that $\sigma = \tau\theta$.

Let A and B be event structures with polarity. A *pre-strategy from* A *to* B is a pre-strategy in $A^\perp \| B$. Write $\sigma : A \relbar\joinrel\mapstochar\joinrel\rightarrow B$ to express that σ is a pre-strategy from A to B. Note that a pre-strategy σ *in* a game A, *e.g.* $\sigma : S \to A$, coincides with a pre-strategy *from* the empty game $\varnothing$ *to* the game A, *i.e.* $\sigma : \varnothing \relbar\joinrel\mapstochar\joinrel\rightarrow A$.

Strategies are defined to be those pre-strategies for which copy-cat behaves as identity w.r.t composition, as we now explain.

4.1.3 Composing pre-strategies

We can present the composition of pre-strategies via pullbacks. Given two pre-strategies $\sigma : S \to A^\perp \| B$ and $\tau : T \to B^\perp \| C$, ignoring polarities we can consider the maps on the underlying event structures, *viz.* $\sigma : S \to A \| B$ and $\tau : T \to B \| C$. Viewed this way we can form the pullback in the category of event structures as shown

$$\begin{tikzcd}
& P \arrow[dl, "\pi_1"'] \arrow[dr, "\pi_2"] \arrow[dd, phantom, "\vee" very near start] & \\
S\|C \arrow[dr, "\sigma\|\mathrm{id}_C"'] & & A\|T \arrow[dl, "\mathrm{id}_A\|\tau"] \\
& A\|B\|C \arrow[d] & \\
& A\|C\,, &
\end{tikzcd}$$

where the map $A\|B\|C \to A\|C$ is undefined on B and acts as identity on A and C. The partial map from P to $A\|C$ given by the diagram above (either way round the pullback square) factors as the composition of the partial map $P \to P \downarrow V$, where V is the set of events of P at which the map $P \to A\|C$ is defined, and a total map $P \downarrow V \to A\|C$. The resulting total map gives us the composition $\tau \odot \sigma : T \odot S =_{\mathrm{def}} P \downarrow V \to A^\perp \| C$ once we reinstate polarities.

In $T \odot S$ we have hidden the synchronization events over B due to the instantiation of Opponent moves of T in B by Player moves of S, and *vice versa.* Later we shall also be concerned with the event structure P, composition before hiding, which we shall denote more descriptively by $T \ast S$.

4.1.4 Concurrent copy-cat

The copy-cat strategy from A to A is an instance of a pre-strategy, so a total map $\gamma_A : \mathbb{C}\!\mathbb{C}_A \to A^\perp \| A$. It is based on the idea that Player moves, of +ve polarity, always copy previous corresponding moves of Opponent, of −ve polarity. For $c \in A^\perp \| A$ we use $\overline{c}$ to mean the corresponding copy of c, of opposite polarity, in the alternative component. Define $\mathbb{C}\!\mathbb{C}_A$ to comprise the event structure with polarity $A^\perp \| A$ together with the extra causal dependencies generated by $\overline{c} \leq_{\mathbb{C}\!\mathbb{C}_A} c$ for all events c with $pol_{A^\perp \| A}(c) = +$. The *copy-cat* pre-strategy $\gamma_A : A \relbar\joinrel\mapstochar\joinrel\rightarrow A$ is defined to be the map $\gamma_A : \mathbb{C}\!\mathbb{C}_A \to A^\perp \| A$ where γ_A is the identity on the common set of events.

4.1.5 Strategies

The main result of [12] is that two conditions on pre-strategies, *receptivity* and *innocence*, are necessary and sufficient for copy-cat to behave as identity w.r.t. the composition of pre-strategies. Receptivity ensures an openness to all possible moves of Opponent. Innocence restricts the behaviour of Player; Player may only introduce new relations of immediate causality of the form $\ominus \rightarrowtriangle \oplus$ beyond those imposed by the game. A pre-strategy σ is *receptive* iff $\sigma x \overset{a}{\longrightarrow\!\!\!\!\subset}$ & $pol_A(a) = - \Rightarrow \exists! s \in S.\ x \overset{s}{\longrightarrow\!\!\!\!\subset}$ & $\sigma(s) = a$. It is *innocent* iff $s \rightarrowtriangle s'$ & $(pol(s) = +$ or $pol(s') = -)$ implies $\sigma(s) \rightarrowtriangle \sigma(s')$. The main result of [12] is that $\gamma_B \odot \sigma \odot \gamma_A \cong \sigma$ iff σ is receptive and innocent. Copy-cats $\gamma_A : A \relbar\!\!\!\!+\!\!\!\!\twoheadrightarrow A$ are receptive and innocent.

A *strategy* is a pre-strategy which is receptive and innocent. We obtain a bicategory in which the objects are event structures with polarity—the games, the arrows from A to B are strategies $\sigma : A \relbar\!\!\!\!+\!\!\!\!\twoheadrightarrow B$ and 2-cells are total maps of pre-strategies with vertical composition the usual composition of such maps. Horizontal composition is given by the composition of strategies $\odot$.

An event structure with polarity S is *deterministic* iff any down-closed set of moves is consistent when its subset of Opponent moves is consistent. Say a strategy $\sigma : S \to A$ is deterministic if S is deterministic. Copy-cat strategies γ_A are deterministic iff the game A is

race-free: for all $x \in \mathcal{C}(A)$ such that $x \overset{a}{\longrightarrow\!\!\!\!\subset}$ and $x \overset{a'}{\longrightarrow\!\!\!\!\subset}$ with $pol(a) = -$ and $pol(a') = +$, we have $x \cup \{a, a'\} \in \mathcal{C}(A)$.

We obtain a sub-bicategory of deterministic strategies between race-free games—in fact equivalent to an order-enriched category [12].

Strategies inherit a duality from pre-strategies. A pre-strategy $\sigma : A \relbar\!\!\!\!+\!\!\!\!\twoheadrightarrow B$ corresponds to a dual pre-strategy $\sigma^\perp : B^\perp \relbar\!\!\!\!+\!\!\!\!\twoheadrightarrow A^\perp$, arising from the correspondence between pre-strategies $\sigma : S \to A^\perp \| B$ and $\sigma^\perp : S \to (B^\perp)^\perp \| A^\perp$.

4.2 Probabilistic strategies

Without information about the stochastic rates of Player and Opponent we cannot hope to ascribe probabilities to outcomes of play in the presence of races, *i.e.* immediate conficts between moves of opposite polarities. Our results on probabilistic strategies depend on restricting to games which are race-free.

It will be convenient to define a probabilistic event structure in which some events are distinguished as Opponent events (where the other events may be Player events or “neutral” events due to synchronizations between Player and Opponent moves). Events which are not Opponent events we shall call *p*-events. For configurations x, y we shall write $x \subseteq^p y$ if $x \subseteq y$ and $y \setminus x$ contains no Opponent events; we write $x \longrightarrow\!\!\!\!\subset^p y$ when $x \longrightarrow\!\!\!\!\subset y$ and $x \subseteq^p y$; we similarly write *e.g.* $x \subseteq^- y$, respectively $x \subseteq^+ y$, if $x \subseteq y$ and $y \setminus x$ comprises solely Opponent, respectively Player, events. We can now extend the notion of configuration-valuation to the situation where events carry polarities.

Definition 4.1 Let E be an event structure in which a specified subset of events are Opponent events. A *configuration-valuation* on E is a function $v : \mathcal{C}(E) \to [0,1]$

for which $v(\varnothing) = 1$,

$$x \subseteq^- y \implies v(x) = v(y) \tag{1}$$

for all $x, y \in \mathcal{C}(E)$, and satisfies the "drop condition"

$$d_v^{(n)}[y; x_1, \cdots, x_n] \geq 0 \tag{2}$$

for all $n \in \omega$ and $y, x_1, \cdots, x_n \in \mathcal{C}(E)$ with $y \subseteq^p x_1, \cdots, x_n$.

A *probabilistic event structure with polarity* comprises E an event structure with polarity together with a configuration-valuation $v : \mathcal{C}(E) \to [0,1]$.

As earlier, by Corollary 3.3, it suffices to verify the "drop condition" for p-covering intervals.

Definition 4.2 Let A be a race-free event structure with polarity. A *probabilistic strategy* v, σ *in* A comprises S, v, a probabilistic event structure with polarity, and a strategy $\sigma : S \to A$. [It follows that S will also be race-free.]

Let A and B be a race-free event structures with polarity. A *probabilistic strategy from* A *to* B is a probabilistic strategy in $A^\perp \| B$.

We extend the usual composition of strategies to probabilistic strategies. Assume probabilistic strategies $\sigma : S \to A^\perp \| B$, with configuration-valuation $v_S : \mathcal{C}(S) \to [0,1]$, and $\tau : T \to B^\perp \| C$ with configuration-valuation $v_T : \mathcal{C}(T) \to [0,1]$. We first define their composition before hiding, as the probabilistic event structure $T * S, v$, tentatively taking $v : \mathcal{C}(T*S) \to [0,1]$ to be $v(x) = v_S(\pi_1 x) \times v_T(\pi_2 x)$ for $x \in \mathcal{C}(T*S)$. This is a configuration-valuation because:

Lemma 4.3 *Let* $v : \mathcal{C}(T*S) \to [0,1]$ *be defined as above. Then,* $v(\varnothing) = 0$*. If* $x \subseteq^- y$ *in* $\mathcal{C}(T*S)$ *then* $v(x) = v(y)$*. Let* $y, x_1, \cdots, x_n \in \mathcal{C}(T*S)$ *with* $y \longrightarrow\!\subset^p x_1, \cdots, x_n$*. Assume that* $\pi_1 y \longrightarrow\!\subset^+ \pi_1 x_i$ *when* $1 \leq i \leq m$ *and* $\pi_2 y \longrightarrow\!\subset^+ \pi_2 x_i$ *when* $m+1 \leq i \leq n$*. Then in* $\mathcal{C}(T*S), v$*,*

$$d_v^{(n)}[y; x_1, \cdots, x_n] = d_{v_S}^{(m)}[\pi_1 y; \pi_1 x_1, \cdots, \pi_1 x_m] \times d_{v_T}^{(n-m)}[\pi_2 y; \pi_2 x_{m+1}, \cdots, \pi_2 x_n]\,.$$

Hence the drop function for v being non-negative is reduced to the drop functions for v_S and v_T being non-negative.

Corollary 4.4 *The assignment* $v(x) = v_S(\pi_1 x) \times v_T(\pi_2 x)$ *to* $x \in \mathcal{C}(T*S)$ *yields a configuration-valuation on* $T*S$*, so a probabilistic event structure* $T*S, v$*.*

We can now complete the definition of the composition of probabilistic strategies. Note that for $x \in \mathcal{C}(T \odot S)$ its down-closure $[x] \in \mathcal{C}(T*S)$.

Lemma 4.5 *Let* A*,* B *and* C *be race-free event structure with polarity. Let* $\sigma : S \to A^\perp \| B$*, with configuration-valuation* $v_S : \mathcal{C}(S) \to [0,1]$*, and* $\tau : T \to B^\perp \| C$ *with configuration-valuation* $v_T : \mathcal{C}(T) \to [0,1]$ *be probabilistic strategies. Assigning* $v_S(\pi_1[x]) \times v_T(\pi_2[x])$ *to* $x \in \mathcal{C}(T \odot S)$ *yields a configuration-valuation which with* $\tau \odot \sigma : T \odot S \to A^\perp \| C$ *forms a probabilistic strategy from* A *to* C*.*

The assumption that games are race-free is needed for Corollary 4.4 and Lemmas 4.3, 4.5. Recall that race-freedom of a game A ensures $\mathrm{C\!C}_A$ is deterministic [12] and hence its copy-cat strategy is easily turned into a probabilistic strategy, as is any deterministic strategy:

Lemma 4.6 *Let S be a deterministic event structure with polarity. Defining $v_S : \mathcal{C}(S) \to [0,1]$ to satisfy $v_S(x) = 1$ for all $x \in \mathcal{C}(S)$, we obtain a probabilistic event structure with polarity.*

Corollary 4.7 *Let A be a race-free game. The copy-cat strategy from A to A comprising $\gamma_A : \mathbb{C}\mathbb{C}_A \to A^\perp \| A$ with configuration-valuation $v_{\mathbb{C}\mathbb{C}_A} : \mathcal{C}(\mathbb{C}\mathbb{C}_A) \to [0,1]$ satisfying $v_{\mathbb{C}\mathbb{C}_A}(x) = 1$, for all $x \in \mathcal{C}(\mathbb{C}\mathbb{C}_A)$, forms a probabilistic strategy.*

Combining the results of this section:

Theorem 4.8 *Race-free games with probabilistic strategies with composition and copy-cat defined as in Lemma 4.5 and Corollary 4.7 inherit the structure of a bicategory from that of games with strategies.*

5 A language of probabilistic strategies

As an indication of the expressivity of probabilistic strategies we show how they straightforwardly include a simple language of probabilistic processes, reminiscent of a higher-order CCS. For this section only, write $\sigma : A$ to mean σ is a probabilistic strategy in game A. Probabilistic strategies are closed under the following operations.

Composition $\tau \odot \sigma : A \| C$, if $\sigma : A \| B$ and $\tau : B^\perp \| C$. Hiding is automatic in a synchronized composition directly based on the composition of strategies.

Simple parallel composition $\sigma \| \tau : A \| B$, if $\sigma : A$ and $\tau : B$—a special case of synchronized composition via the identification of $\sigma \| \tau$ with $\tau \odot \sigma$, in which $\sigma : A^\perp \mathrel{\longrightarrow\!\!\!\!\!\!+\;\,} \varnothing$ and $\tau : \varnothing \mathrel{\longrightarrow\!\!\!\!\!\!+\;\,} B$.

Input prefixing $\sum_{i \in I} \ominus.\sigma_i : \sum_{i \in I} \ominus.A_i$, if $\sigma_i : A_i$, for $i \in I$, where I is countable.

Output prefixing $\sum_{i \in I} p_i \oplus .\sigma_i : \sum_{i \in I} \oplus.A_i$, if $\sigma_i : A_i$, for $i \in I$, where I is countable, and $p_i \in [0,1]$ for $i \in I$ with $\sum_{i \in I} p_i \le 1$. If $\sum_{i \in I} p_i < 1$, there is non-zero probability of terminating without any action. By design $(\sum_{i \in I} \oplus.A_i)^\perp = \sum_{i \in I} \ominus.A_i^\perp$.

Relabelling, the composition $f\sigma : B$, if $\sigma : A$ and $f : A \to B$ is itself a strategy, *i.e.* total, receptive and innocent.

Pullback $f^*\sigma : A$, if $\sigma : B$ and $f : A \to B$ is a map of event structures which preserves +-conflict, *i.e.* is defined on all +ve events and satisfies

$$x \overset{a_1}{-\!\!\!\subset} x_1 \;\&\; x \overset{a_2}{-\!\!\!\subset} x_2 \;\&\; pol(a_1) = pol(a_2) = + \;\&\; x_1 \mathrel{\not\uparrow} x_2 \implies f x_1 \mathrel{\not\uparrow} f x_2 \,.$$

The strategy $f^*\sigma$ is got by the pullback

$$\begin{tikzcd} S' \arrow[r, "f'"] \arrow[d, "f^*\sigma"'] \arrow[dr, phantom, very near start, "\lrcorner"] & S \arrow[d, "\sigma"] \\ A \arrow[r, "f"'] & B \,. \end{tikzcd}$$

Then, the map f' also preserves +-conflict. The configuration valuation $v_{S'}$ of S' is defined from that v_S of S by taking $v_S(x) = v_S(f'x)$, for $x \in \mathcal{C}(f^*S)$. If $\sigma : S \to B$ is a strategy then so is $f^*\sigma : S' \to A$. Pullback along $f : A \to B$ may introduce causal links, present in A but not in B.

Abstraction $\lambda x : A.\sigma : A \multimap B$. Because probabilistic strategies form a monoidal-closed bicategory, with tensor $A \| B$ and function space $A \multimap B =_{\rm def} A^\perp \| B$, they support an (affine) λ-calculus with λ-abstractions, which in this context permits process-passing as in [11].

Recursive probabilistic processes can be dealt with along standard lines [14].

The types as they stand are somewhat inflexible. These limitations can be remedied by introducing monads T and new types of the form $T(A)$, though doing this in sufficient generality would involve the introduction of symmetry to games—see Section 7.

6 Quantum strategies

A more novel application is to a definition of quantum event structures and strategies. Throughout let $\mathcal{H}$ be a separable Hilbert space over the complex numbers. For operators A, B on $\mathcal{H}$ we write $[A, B] =_{\rm def} AB - BA$.

6.1 Quantum event structures

Definition 6.1 A *quantum event structure* (over $\mathcal{H}$) comprises an event structure $(E, \leq, \mathrm{Con})$ together with an assignment Q_e of projection or unitary operators on $\mathcal{H}$ to events $e \in E$ such that for all $e_1, e_2 \in E$,

$$e_1 \; co \; e_2 \implies [Q_{e_1}, Q_{e_2}] = 0\,.$$

Given a finite configuration, $x \in \mathcal{C}(E)$, define the operator A_x to be the composition $Q_{e_n} Q_{e_{n-1}} \cdots Q_{e_2} Q_{e_1}$ for some covering chain

$$\varnothing \overset{e_1}{\longrightarrow\!\!\!\subset} x_1 \overset{e_2}{\longrightarrow\!\!\!\subset} x_2 \cdots \overset{e_n}{\longrightarrow\!\!\!\subset} x_n = x$$

in $\mathcal{C}(E)$. This is well-defined as for any two covering chains up to x the sequences of events are Mazurkiewicz trace equivalent, *i.e.* obtainable, one from the other, by successively interchanging concurrent events. In particular $A_\varnothing$ is the identity operator on $\mathcal{H}$.

An *initial state* is given by a density operator ρ on $\mathcal{H}$.

We regard $w \in \mathcal{C}^\infty(E)$ as a partial quantum experiment—it is 'partial' in the sense that it might extend to $w' \supseteq w$ in $\mathcal{C}^\infty(E)$. An experiment w specifies which unitary operators (events of preparation) and projection operators (elementary positive tests) to apply and in which order. The order being partial permits commuting operators to be applied concurrently, independently of each other, perhaps in a distributed fashion.

Consider a quantum event structure with initial state ρ. While it does not make sense to attribute a probability distribution globally, over the whole space of configurations $\mathcal{C}^\infty(E)$, the next theorem says that with respect to any experiment w there is a probability distribution q_w over its possible outcomes. (Below, by an unnormalized density operator we mean a positive, self-adjoint operator with trace less than or equal to one.)

Theorem 6.2 *Let E, Q be a quantum event structure with initial state ρ. Each configuration $x \in \mathcal{C}(E)$ is associated with an unnormalized density operator $\rho_x =_{\rm def} A_x \rho A_x^\dagger$ and a value in $[0,1]$ given by $v(x) =_{\rm def} \mathrm{Tr}(\rho_x) = \mathrm{Tr}(A_x^\dagger A_x \rho)$. For any $w \in \mathcal{C}^\infty(E)$, the function v restricts to a configuration-valuation v_w on the elementary event structure w (viz. the event structure with events w, and causal dependency and (trivial) consistency inherited from E); hence v_w extends to a probability measure q_w on $\mathcal{F}_w =_{\rm def} \{x \in \mathcal{C}^\infty(E) \mid x \subseteq w\}$.*

6.2 Quantum strategies

A *quantum game* comprises A, pol, Q, ρ where A, pol is a race-free event structure with polarity, A, Q is a quantum event structure with initial state ρ. A *strategy* in the quantum game comprises a probabilistic strategy in A, so a strategy $\sigma : S \to A$ together with configuration-valuation v on S.

Given a strategy $v_S, \sigma : S \to A$ and counter-strategy $v_T, \tau : T \to A^\perp$ in a quantum game A, Q, ρ we obtain as their composition before hiding the probabilistic event structure $T \ast S$ with configuration-valuation $v(x) =_{\rm def} v_S \pi_1(x) \times v_T \pi_2(x)$ on $x \in \mathcal{C}(T \ast S)$—see Corollary 4.4. The event structure $T \ast S$ is obtained as a pullback—Section 4.1.3—and is associated with a map $f =_{\rm def} \sigma\pi_1 = \tau\pi_2 : T \ast S \to A$. We can interpret $f : T \ast S \to A$ as the probabilistic experiment which results from the interaction of the strategy σ and the counter-strategy τ. The event structure $T \ast S$ carries a probability measure μ_v. The probability that the play-off of σ against τ produces a result in a Borel subset U of of $\mathcal{C}^\infty(A)$, is given by the Lebesgue integral

$$\int q_w(U \cap \mathcal{F}_w)\, d\mu_v f^{-1}(w)\,.$$

Strategies in quantum games inherit the types and language of probabilistic strategies, though additional constructs will be needed to introduce entanglement beyond that already present in a given start state.

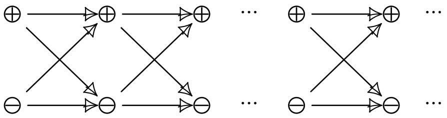
7 Extensions

As they stand the games here are games of *perfect information*. In games of *imperfect information* some moves are masked, or inaccessible, and strategies with dependencies on unseen moves are ruled out. It is straightforward to extend concurrent games to games with imperfect information in way that respects the operations of the bicategory of games [17] and does not disturb the addition of probability. A fixed preorder of *levels* $(\Lambda, \leq)$ is pre-supposed. The levels are to be thought of as levels of access, or permission. A Λ-game comprises a game A with a *level function* $l : A \to \Lambda$ such that if $a \leq_A a'$ then $l(a) \leq l(a')$ for all $a, a' \in A$. A probabilistic Λ-strategy in the Λ-game is a probabilistic strategy $v_S, \sigma : S \to A$ for which if $s \leq_S s'$ then $l\sigma(s) \leq l\sigma(s')$ for all $s, s' \in S$. One interpretation of Λ, pertinent to the treatment of quantum strategies, is as space-time with $\lambda \leq \lambda'$ meaning there is a causal curve from λ to λ'.

We can add *payoff* to a game A as a Borel measurable function $X : \mathcal{C}^\infty(A) \to \mathbb{R}$. Given a probabilistic strategy $v_S, \sigma : S \to A$ and counter-strategy $v_T, \tau : T \to A^\perp$ we obtain their composition before hiding as their pullback $T \ast S, \pi_1, \pi_2$, associated with the map $f =_{\mathrm{def}} \sigma\pi_1 = \tau\pi_2 : T * S \to A$. The event structure $T * S$ comes equipped with a configuration-valuation $v(x) = v_S(\pi_1 x) \times v_T(\pi_2 x)$, for $x \in \mathcal{C}(T * S)$. The *expected payoff* is obtained as the Lebesgue integral

$$\mathbf{E}_{\sigma,\tau}(X) = \int X(fx)\ d\mu_{\sigma,\tau}(x)\,.$$

In particular, *Blackwell games* [10] become a special case of probabilistic Λ-games with payoff. For Blackwell games an appropriate choice of Λ is the infinite elementary event structure:



A Blackwell game is given by A, a race-free concurrent game with payoff X, for which there is a (necessarily unique) polarity-preserving rigid map from A to Λ—this map becomes the level function. Moves in A occur in rounds comprising a choice of move for Opponent and a choice of move for Player made concurrently. The existing literature is often concerned with *total* strategies which always progress, which we can express in our general context by insisting non $\subseteq^{+}$-maximal finite configurations of the strategy are transient—*cf.* Example 3.7. Traditionally, in Blackwell games a strategy (for Player) is a total Λ-strategy in such a Λ-game—strategies are restricted to those assigning *total* probability distributions at each round.

There are several reasons to consider symmetry in games, situations where distinct plays are essentially similar to one another. Symmetry can help in the analysis of games, by for instance reducing the number of cases to consider. Symmetry can also help compensate for the overly-concrete nature of event structures in representing games; many useful operations on games which are not monads or comonads w.r.t. strategies become so *up to symmetry* [15,2] and this leads, for example, to richer type systems. Symmetry on an event structure can be captured through an *isomorphism family* which expresses when one finite configuration of the event structure is essentially the same as another [15]. It is a straightforward matter to ensure that configuration-valuations, attributing probability, respect the isomorphism family. The addition of symmetry to games meshes well with the introduction of probability. This should enable a formal connection with the probabilistic games of Danos and Harmer [3] which are based on HO games—allowing copying, so whose relation with concurrent games requires suitable (co)monads to exist, so symmetry.

Acknowledgments Discussion with Samson Abramsky, Nathan Bowler, Peter Hines, Ohad Kammar, Mike Mislove and Prakash Panangaden has been helpful. Daniele Varacca deserves special thanks for our earlier work on probabilistic event structures. I gratefully acknowledge the ERC Grant ECSYM.

References

[1] Samy Abbes and Albert Benveniste. True-concurrency probabilistic models: Branching cells and distributed probabilities for event structures. *Inf. Comput.*, 204(2):231–274, 2006.

[2] Simon Castellan, Pierre Clairambault, and Glynn Winskel. Symmetry in concurrent games. Submitted, 2013.

[3] Vincent Danos and Russell Harmer. Probabilistic game semantics. In *LICS'00*. IEEE Computer Society, 2000.

[4] Yannick Delbecque. Game semantics for quantum data. *QPL/DCM 2008, Electr. Notes Theor. Comput. Sci.*, 2008.

[5] J.O. Grabbe. An introduction to quantum game theory. *arXiv preprint quant-ph/0506219*, 2005.

[6] Robert B. Griffiths. *Consistent quantum theory.* CUP, 2002.

[7] Claire Jones and Gordon Plotkin. A probabilistic powerdomain of valuations. In *LICS '89*. IEEE Computer Society, 1989.

[8] Joost-Pieter Katoen. *Quantitative and Qualitative Extensions of Event Structures.* PhD Thesis, University of Twente, 1996.

[9] N Saheb-Djahromi M Alvarez-Manilla, A Edalat. An extension result for continuous valuations. *Journal of the London Mathematical Society*, 61(2):629–640, 2000.

[10] Donald Martin. The determinacy of Blackwell games. *Journal of Symbolic Logic*, 63(4):1565–1581, 1998.

[11] Mikkel Nygaard and Glynn Winskel. Linearity in process languages. In *LICS'02*. IEEE Computer Society, 2002.

[12] Silvain Rideau and Glynn Winskel. Concurrent strategies. In *LICS 2011*. IEEE Computer Society, 2011.

[13] Daniele Varacca, Hagen Völzer, and Glynn Winskel. Probabilistic event structures and domains. *Theor. Comput. Sci. 358(2-3): 173-199*, 2006.

[14] Glynn Winskel. Event structure semantics for CCS and related languages. In *ICALP'82*, volume 140 of *LNCS*. Springer, 1982.

[15] Glynn Winskel. Event structures with symmetry. *Electr. Notes Theor. Comput. Sci. 172: 611-652*, 2007.

[16] Glynn Winskel. Event structures, stable families and games. Lecture Notes, Cambridge University Computer Laboratory: http://www.cl.cam.ac.uk/~gw104/EvStrsStFamGames.pdf, 2012.

[17] Glynn Winskel. Winning, losing and drawing in concurrent games with perfect or imperfect information. In *Festschrift for Dexter Kozen*, volume 7230 of *LNCS*. Springer, 2012.

Approximating Bisimilarity for Markov Processes[1]

Chunlai Zhou[2]

Computer Science Department
School of Information
Renmin University of China
Beijing, CHINA

Abstract

In this paper we investigate bisimilarity for *general* Markov processes through the correspondence between sub-σ-algebras and equivalence relations. In particular, we study bisimulations from the perspective of fixed-point theory. Given a Markov process $M = \langle \Omega, \Sigma, \tau \rangle$, we characterize its state bisimilarity as the *greatest* fixed point of a composition of two natural set operators between equivalence relations on Ω and sub-σ-algebras of Σ. Moreover, we employ a Smith-Volterra-Cantor-set-construction to obtain an example to show that state bisimilarity is beyond ω iterations of these two operators alternately from event bisimilarity and hence the composite operator is not continuous. This process of iteration illustrates the gap between event bisimilarity (or logical equivalence) and state bisimilarity, and hence provides insights about the Hennessy-Milner property for general Markov processes. At the end of this paper, we also study approximation of Markov processes related to filtration.

Keywords: Markov processes, state bisimilarity, event bisimilarity, fixed point, Hennessy-Milner logic

1 Introduction

Markov processes with continuous state spaces are important mathematical models in different physical sciences such as physics, biology, finance and computer sciences. The dynamics of the processes is governed by the present state rather than by the past history of the processes. With the ever-growing computer technology, we need to develop a theory of computational grip of this kind of important structures. If one is interested in computing them, we must build a machinery to approximate Markov processes with continuous state space and also make sure that the approximating processes *preserve* all the essential properties especially the dynamic aspects of the original processes.

[1] I want to thank Professor Ernst-Erich Doberkat, Ingo Battenfeld, Yuan Feng and Pedro Sánchez Terraf for helpful discussions at different stages of this research. This work is partly supported by NSF of China (Grant Number: 60905036), by the Key Project for basic research from the Ministry of Science and Technology of China (Grant Number: 2012CB316205), by the Basic Research Funds in Renmin University from the Central Government (Grant No. 201230005) and by German Academic Exchange Service (DAAD Codenumber: A/10/00803).

[2] Email: chunlai.zhou@gmail.com

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.com/locate/entcs

The limit of the approximating processes is usually not the original approximated process but instead the *quotient* process with respect to some *bisimilarity.* There are two totally different notions of bisimilarities for Markov processes in the literature. The first one is called *state bisimilarity.* Intuitively, two states are state bisimilar in the process if they match transition probabilities for the same moves. And the other one is called *event bisimilarity.* Two states are event bisimilar if they are indistinguishable by any sub-σ-algebra of events that respects the dynamics in the process. For any *general* Markov process, event bisimilarity coincides with logical equivalence and is a superset of state bisimilarity [4]. For Markov processes on analytical spaces or Polish spaces [5][7], these three kinds of equivalences are the same, which is the well-known Hennessy-Milner property. However, a general Markov process does not necessarily satisfy the Hennessy-Milner property [12].

Conceptually, there is a *mismatch* between approximating Markov processes and bisimilarities in the literature. Most approaches to approximate Markov processes [6][15][3][13] employ similar *syntactic* machineries. The limit of the approximating Markov processes is the quotient Markov process with respect to event bisimilarity (or logical equivalence). However, it is the quotient Markov process with respect to *state bisimilarity* that preserves the dynamics of the original Markov process.

In order to understand better the approximation of Markov processes, we study in this paper approximating bisimilarity for *general* Markov processes. There are two approaches for approximating bisimilaries: bottom-up and top-down. The approximation according to the bottom-up approach is essentially syntactic and consists of a sequence of n-bisimilarities, which corresponds to logical equivalence up to depth n. So this approach is about event bisimilarity and is in spirit closely related to those of approximating Markov process in the literature. The second and top-down approach is semantical and studies state bisimilarity from the perspective of fixed-point theory. Given a Markov process $M = \langle \Omega, \Sigma, \tau \rangle$, we characterize its state bisimilarity as the *greatest* fixed point of a composition $\mathbf{O}$ of two natural set operators between equivalence relations on Ω and sub-σ-algebras of Σ (Section 4). Not only may state bisimilarity be obtained from the universal relation on Ω by iterating α times the composite operator $\mathbf{O}$ for some ordinal α, but also it can be reached top-down from event bisimilarity by iterating β times $\mathbf{O}$ for some ordinal β. This top-down approach is actually reflected in many algorithms of computing bisimilarity in the literature [5] [6]. In this paper, we employ the above ordinal β to *measure* the gap from event bisimilarity to state bisimilarity. Sánchez Terraf [12] constructed an example and showed that the gap there is at least one. In this paper, we employ a Smith-Volterra-Cantor set (so-called *fat* Cantor set) to build an example and show that the gap is beyond the limit ordinal ω. This implies that the operator $\mathbf{O}$ is not continuous and the gap between state and event bisimilarities is *very big.* Also the example illustrates the gap between the above two approaches for approximating bisimilarities: bottom-up and top-down.

At the end of the paper, we present a general theory about filtration as an approach of approximating Markov processes and discuss its relations to the above approaches to approximate bisimilarities. In particular we provide another characterization of the Hennessy-Milner property through *filtration.* Essentially, a filtration of a Markov process M' through a sublanguage $\mathcal{L}'$ of the whole language $\mathcal{L}$

for Markov processes is its quotient that respects the satisfiability of all formulas in $\mathcal{L}'$. We show (Theorem 5.4) that a Markov process satisfies the Hennessy-Milner property iff it has *only one* filtration through the language $\mathcal{L}$.

2 Preliminaries

Let $\mathcal{A}$ be a (Boolean) algebra on a set X, i.e. a non-empty collection of subsets of X closed under complements and binary unions. $\mathcal{A}$ is a σ-algebra if it is also closed under countable unions. If $\mathcal{A}$ is a σ-algebra, then $\mathcal{X} = \langle X, \mathcal{A}\rangle$ is a *measurable space* and the elements of $\mathcal{A}$ are usually called *events* or *measurable subsets of* X. We write $\sigma(\mathcal{A}_0)$ for the smallest σ-algebra containing a given set $\mathcal{A}_0$ of subsets of $\mathcal{A}$. When $\sigma(\mathcal{A}_0) = \mathcal{A}$, we usually say that $\mathcal{A}_0$ *generates* $\mathcal{A}$. A *measurable function* $f : \langle X, \mathcal{A}\rangle \to \langle X', \mathcal{A}'\rangle$ is a function $f : X \to X'$ such that, for any $A' \in \mathcal{A}', f^{-1}(A') \in \mathcal{A}$ where $\langle X', \mathcal{A}'\rangle$ is also a measurable space. A set function $\mu : \mathcal{A} \to [0, \infty]$ on $\mathcal{A}$ in X is *finitely additive* if $\mu(A_1 \cup A_2) = \mu(A_1) + \mu(A_2)$ whenever A_1 and A_2 are disjoint elements of $\mathcal{A}$. μ is called a (countably additive)*measure* if it satisfies the following conditions:

(i) $\mu(\emptyset) = 0$;

(ii) $\mu(\bigcup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} \mu(A_i)$ where $\{A_i\}_{i=1}^{\infty}$ is a pairwise disjoint sequence of events of $\mathcal{A}$.

The second property is usually called the *countable additivity.* The measure μ is *finite* or *infinite* as $\mu(X) < \infty$ or $\mu(X) = \infty$. If $\mu(X) \leq 1$, then μ is called a *subprobability measure.* If $\mu(X) = 1$, then μ is called a *probability measure.* A metric space $\langle X, \rho\rangle$ is *complete* if any Cauchy sequence has a limit in X, and ρ is called a *complete* metric. A topological space $\langle X, \tau\rangle$ is called *separable* if it has a countable dense subset. A *Polish space* $\langle X, \tau\rangle$ is a separable topological space which is metrizable through a complete metric. The *Borel σ-algebra* $\mathcal{B}(X, \tau)$ for the topology τ is the smallest σ-algebra that contains τ. An *analytical* space is the image of a Polish space under a continuous function from one Polish space to another. The interested reader may refer to [1] for the basics about measure theory.

A *transition (sub)probability function* T on a measurable space $\mathcal{X} = \langle X, \mathcal{A}\rangle$ is a function from $X \times \mathcal{A}$ to $[0, 1]$ satisfying the following two conditions:

- for each $x \in X$, $T(x, \cdot)$ is a (sub)probability measure, and
- for each $A \in \mathcal{A}$, $T(\cdot, A)$ is a measurable function.

T is also called a *Markov kernel.* A *Markov process* M is a structure $\langle X, \mathcal{A}, T\rangle$, where $\langle X, \mathcal{A}\rangle$ is measurable space and T is a subprobability transition function. A function $f : \langle X, \mathcal{A}, T\rangle \to \langle X', \mathcal{A}', T'\rangle$ is a *zigzag morphism* if it is surjective, measurable, and the following equality holds:

$$T(x, f^{-1}(A')) = T'(f(x), A'), \text{ for any } x \in X, A' \in \mathcal{A}'.$$

The two Markov processes $\langle X, \mathcal{A}, T\rangle$ and $\langle X', \mathcal{A}', T'\rangle$ are probabilistically *bisimilar* if there is a Markov process $\langle X'', \mathcal{A}'', T''\rangle$ with two *surjective* zigzag morphisms $h' : X'' \to X'$, and $h : X'' \to X$.

One important result about Markov processes is that there is a Hennessy-Milner logic to characterize the above probabilistic bisimulation. A *formula* ϕ of the logic is formed by the following sytax:

$$\phi := \top \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid L_r\phi (r \in \mathbb{Q} \cap [0,1])$$

where $\mathbb{Q}$ is the field of rationals. $\mathcal{L}$ denotes the language of this simple syntax. The *depth* $dp(\phi)$ of formulas ϕ is defined inductively as in modal logic. $\mathcal{L}_n$ denotes the sublanguage of $\mathcal{L}$ of formulas of depeth $\leq n$ $(n = 0, 1, 2, \cdots)$. The interpretation of formulas in the Markov process $M = \langle S, \mathcal{A}, T\rangle$ is straightforward except the following crucial clause:

$$M, w \models L_r\phi \text{ iff } T(w)([\![\phi]\!]_M) \geq r, \text{ where } [\![\phi]\!]_M := \{w \in S : M, w \models \phi\}.$$

A formula ϕ is called satisfied at w in M if $M, w \models \phi$. Two states in S are called *logical equivalent* if they satisfy the same set of formulas in $\mathcal{L}$. The following is the well-known theorem about the Hennessy-Milner property or expressivity of Markov processes [5][7][8].

Theorem 2.1 *Let $\langle S, \mathcal{A}, T\rangle$ be a Markov process in which S is a Polish space and $\mathcal{A}$ is a Borel σ-algebra. Two states are probabilistically bisimilar iff they satisfy the same set of formulas of $\mathcal{L}$.*

Before moving to the main part, we first fix some notations. Let $\langle \Omega, \Sigma\rangle$ be a measurable space and R be an equivalence relation on Ω. For $E \subseteq \Omega$, $E/_R$ denotes the set $\{[s]_R : s \in E\}$ and, for $\Sigma' \subseteq \Sigma$, $\Sigma'/_R = \{E'/_R : E' \in \Sigma'\}$. It is easy to see that $\langle \Omega/_R, \Sigma/_R\rangle$ is also a measurable space. Conversely, for $B \subseteq \Omega/_R$, $B_\cup$ denotes $\bigcup B$ and, for $\Sigma'' \subseteq \Sigma/_R$, $\Sigma''_\cup$ denotes the set $\{B_\cup : B \in \Sigma''\}$. In particular, $2_\cup^{\Omega/R}$ denotes the set $\{A \in 2^\Omega : A = \bigcup C \text{ for some } C \in 2^{\Omega/R}\}$ [3]. For the equivalence relation R on Ω, elements of Σ are called *R-closed* if they are also unions of R-equivalence classes. $\Sigma(R)$ denotes the sub-σ-algebra of R-closed events in Σ, i.e., $\Sigma \cap 2_\cup^{\Omega/_R}$. Let $\Sigma(\cdot)$ denote this mapping from equivalence relations on Ω to sub-σ-algebras of Σ. Conversely, for any sub-σ-algebra Σ' of Σ, $\mathbf{R}(\Sigma')$ denotes the equivalence relation:

$$s\mathbf{R}(\Sigma')s' \text{ if, for any } A' \in \Sigma' \ (s \in A' \Leftrightarrow s' \in A').$$

Let $\mathbf{R}(\cdot)$ denote this mapping from sub-σ-algebras of Σ to equivalence relations on Ω. It is easy to check that, given the space (Ω, Σ), these two maps $\Sigma(\cdot)$ and $\mathbf{R}(\cdot)$ form a Galois connection [4]:

(i) for any sub-σ-algebra $\mathcal{B}$ of Σ, $\mathcal{B} \subseteq \Sigma(\mathbf{R}(\mathcal{B}))$;

(ii) for any equivalence relation R' on S, $R' \subseteq \mathbf{R}(\Sigma(R'))$.

For a Markov process $M = \langle \Omega, \Sigma, \tau\rangle$ on $\langle \Omega, \Sigma\rangle$, we define a relation $\mathbf{R}^T(M)$ as follows: any s and t in Ω,

$$(s, t) \in \mathbf{R}^T(M) \text{ whenever } \tau(s, E) = \tau(t, E) \text{ for all } E \in \Sigma.$$

Whenever τ is clear, we also write $\mathbf{R}^T(M)$ as $\mathbf{R}^T(\Sigma)$. Let $\mathbf{R}^T(\cdot)$ denote this mapping from sub-σ–algebras of Σ to equivalence relations on Ω. $\mathbf{R}^T(\cdot)$ and $\Sigma(\cdot)$ don't

[3] In topology, it is usualy denoted as $(\Omega/_R)^\cup$.

generally form a Galois connection (Example **??**). Let Σ' be a sub-σ-algebra of Σ. We say that Σ' is *stable* with respect to $M = \langle \Omega, \Sigma, \tau \rangle$ if, for all $E \in \Sigma'$, $r \in [0,1]$,

$$\{w \in \Omega : \tau(w, E) > r\} \in \Sigma'.$$

It is easy to see that Σ' is stable iff $\tau(\cdot, E)$ is Σ'-measurable for each $E \in \Sigma'$,i.e., $\langle \Omega, \Sigma', \tau \rangle$ is a Markov process [4].

Lemma 2.2 *Let R be an equivalence relation on Ω and Σ' be a sub-σ-algebra of Σ such that Σ' is stable.*

(i) *$\langle \Omega, \Sigma(R), \tau \rangle$ is a Markov process if and only if $\tau(\cdot, E)$ is constant on R-classes for all $E \in \Sigma(R)$.*

(ii) *$\boldsymbol{R}(\Sigma') \subseteq \boldsymbol{R}^T(\Sigma')$.*

Note that, generally, Part 2 does not hold if Σ' is not stable.

Lemma 2.3 *Let $\langle \Omega, \Sigma, \tau \rangle$ be a Markov process, Σ_1 and Σ_2 be two sub-σ-algebras of Σ, and R_1 and R_2 be two equivalence relations on Ω.*

(i) *If $\Sigma_2 \subseteq \Sigma_1$, then $\boldsymbol{R}^T(\Sigma_2) \supseteq \boldsymbol{R}^T(\Sigma_1)$ and $\boldsymbol{R}(\Sigma_2) \supseteq \boldsymbol{R}(\Sigma_1)$.*

(ii) *If $R_1 \subseteq R_2$, then $\Sigma(R_1) \supseteq \Sigma(R_2)$.*

(iii) *If $\Sigma_2 \subseteq \Sigma_1$, then $\Sigma(\boldsymbol{R}^T(\Sigma_2)) \subseteq \Sigma(\boldsymbol{R}^T(\Sigma_1))$.*

(iv) *If $R_1 \subseteq R_2$, then $\boldsymbol{R}^T(\Sigma(R_1)) \subseteq \boldsymbol{R}^T(\Sigma(R_2))$.*

3 Fixed-point characterization of state bisimilarity

In the following sections, we consider a given Markov process $M = \langle S, \mathcal{A}, \tau \rangle$ and study relationships between sub-σ-algebras of $\mathcal{A}$ and equivalence relations on S.

Definition 3.1 An equivalence relation R on Markov process $M := \langle S, \mathcal{A}, \tau \rangle$ is called a *state bisimulation* if $R \subseteq \mathbf{R}^T(\mathcal{A}(R))$, namely,

for any $s, t \in S$, sRt implies that $\tau(s, E) = \tau(t, E)$ for every $E \in \mathcal{A}(R)$.

In other words, R is a state bisimulation if it is a *post-fixpoint* of the composite operator $\mathbf{R}^T(\mathcal{A}(\cdot))$. From Part (1) of Lemma 2.2, we know that R is a state bisimulation iff $\langle S, \mathcal{A}(R), \tau \rangle$ is a Markov process. Two sates s and t in S are *state bisimilar* if there is a state bisimulation R such that $(s, t) \in R$. An equivalence relation R' on M is called an *event bisimulation* if it is defined through a Markov process with a sub-σ-algebra $\mathcal{A}'$ in the sense that

- $R' = \mathbf{R}(\mathcal{A}')$, i.e., for any $s, t \in S$, $sR't$ iff s and t are indistinguishable in $\mathcal{A}'$;
- $\langle S, \mathcal{A}', \tau \rangle$ is a Markov process.

Two states s and t are *event bisimilar* if there is an event bisimulation R' such that $(s, t) \in R'$.

The classes of both state and event bisimulations are closed under the following operation: for arbitrary index set I,

- $\bigvee_{i \in I} R_i := (\bigcup_{i \in I} R_i)^*$ where $(\bigcup_{i \in I} R_i)^*$ denotes the transitive closure of the relation $\bigcup_{i \in I} R_i$.

Thus state bisimilarity is the union of all state bisimulations, and event bisimilarity that of state bisimulations. $\approx_M$ and $\sim_M$ denote state and event bisimilarities on M, respectively. When the context is clear, we usually drop the subscript M.

Originally, Danos et. al. [4] would like to present event bisimulation as a *weakening* of state bisimulation. However, from the following example (adapted from Example 4.11 in [4]), we know that a state bisimulation R is not in general an event bisimulation although a closely-related *bigger* state bisimulation $\mathbf{R}(\mathcal{A}(R))$ is indeed an event bisimulation (part 4 of the following proposition, which is from [4]).

Proposition 3.2 *Let R be a state bisimulation.*

(i) $R \subseteq \boldsymbol{R}(\mathcal{A}(R))$*;*

(ii) *If Λ is a sub-σ-algebra of $\mathcal{A}$, $\boldsymbol{R}(\Lambda) = \boldsymbol{R}(\mathcal{A}(\boldsymbol{R}(\Lambda)))$ and $\Lambda \subseteq \mathcal{A}(\boldsymbol{R}(\Lambda))$;*

(iii) *R is an event bisimulation iff $R = \boldsymbol{R}(\mathcal{A}(R))$;*

(iv) *$\boldsymbol{R}(\mathcal{A}(R))$ is both a state bisimulation and an event bisimulation.*

In the remainder of this section, we will investigate state (event) bisimulation from the perspective of *fixed-point* theory. From the above Proposition 3.2, we know that, if an equivalence relation R on S is both a state bisimulation and an event bisimulation, it is a fixed point of the operator $\mathbf{R}(\mathcal{A}(\cdot))$ on the class of equivalence relations on S.

Theorem 3.3 *Both state bisimilarity and event bisimilarity are fixed points of the composite operator $\boldsymbol{R}(\mathcal{A}(\cdot))$. So state bisimilarity $\approx$ is also an event bisimulation and hence $\approx \subseteq \sim$.*

Proof. From Proposition 3.2, we know that $\approx \subseteq \mathbf{R}(\mathcal{A}(\approx))$ and $\mathbf{R}(\mathcal{A}(\approx))$ is a state bisimulation. Since $\approx$ is the union of all state bisimulations, $\approx = \mathbf{R}(\mathcal{A}(\approx))$. The proof for event bisimilarity is similar. □

However, event bisimilarity $\sim$ is not the *greatest* fixed point of the operator, since the universal relation $S \times S$ is also a fixed point. In the next section, we will show that generally the above containment in Theorem 3.3 is strict.

Theorem 3.4 *The state bisimilarity $\approx$ is the greatest fixed point of the composite operator $\boldsymbol{R}^T(\mathcal{A}(\cdot))$.*

Proof. We know from Theorem 3.3 that, for state bisimilarity $\approx$, $\approx = \mathbf{R}(\mathcal{A}(\approx))$. Since $\langle S, \mathcal{A}(\approx), \tau\rangle$ is a Markov process, $\approx = \mathbf{R}(\mathcal{A}(\approx)) \subseteq \mathbf{R}^T(\mathcal{A}(\approx))$ (according to Lemma 3.2). Let R' denote $\mathbf{R}^T(\mathcal{A}(\approx))$. It follows that $\mathcal{A}(R') \subseteq \mathcal{A}(\approx)$ and hence $\tau(\cdot, E)$ is constant on R'-classes for all $E \in \mathcal{A}(\approx)$ and hence is constant on R'-classes for all $E \in \mathcal{A}(R')$. It follows from Lemma 2.2 that R' is a state bisimulation. Since we have shown $\approx \subseteq R'$ and $\approx$ is the greatest state bisimulation, the state bisimilarity $\approx$ is the same as R'. In other wors, $\approx$ is also the fixed point of the operator $\mathbf{R}^T(\mathcal{A}(\cdot))$. It is also the greatest fixed point. Indeed, each fixed point R of $\mathbf{R}^T(\mathcal{A}(\cdot))$ is also a state bisimulation and hence is contained in the state bisimilarity $\approx$.

□

One may also appeal *directly* to the well-known Tarski-Knaster Theorem (Chapter 1 of [11]) to show that state bisimilarity is the greatest fixed point of the composite operator $\mathbf{R}^T(\mathcal{A}(\cdot))$. Desharnais et.al. [6] also studied state bisimilarity from the perspective of fixed point but did not consider its relationship with other bisimilarities. The main purpose of our above presentation of state bisimilarity by detouring to transition bisimilarity is to characterize both the relationships among different bisimulations and the gaps among them through the operator $\mathbf{R}^T(\mathcal{A}(\cdot))$.

4 Gap between state and event bisimilarities

For simplicity, we use $\mathbf{O}$ to denote the composite operator $\mathbf{R}^T(\mathcal{A}(\cdot))$. For a relation R, we construct by transfinite induction a chain of equivalence relations on $M = \langle S, \mathcal{A}, \tau\rangle$ as follows:

- $\mathbf{O}^{\alpha+1}(R) = \mathbf{O}(\mathbf{O}^{\alpha}(R))$;
- $\mathbf{O}^{\lambda}(R) = \bigcup_{\alpha<\lambda} \mathbf{O}^{\alpha}(R)$ if λ is a limit ordinal.

According to Lemma 2.3, $\mathbf{O}$ is monotonic.

Theorem 4.1 *For the above operator* $\mathbf{O}$*,*

(i) *The greatest fixed point exists and is* $\mathbf{O}^{\alpha}(R_u)$ *for some ordinal* α*. So state bisimilarity can be obtained by iterating the operator* $\mathbf{O}$ α *times from the universal relation* R_u *for some ordinal* α *whose cardinality is no larger than that of* S*.*

(ii) *state bisimilarity* $\approx$ *can be obtained from event bisimilarity* $\sim$ *by iterating* α *times* $\mathbf{O}$ *for some ordinal* α*; in other words,* $\approx\; = \mathbf{O}^{\alpha}(\sim)$*.*

Proof. The first part follows trivially from Tarski-Knaster's fixed point Theorem and the second from Theorem 3.4 and Lemma 2.3. □

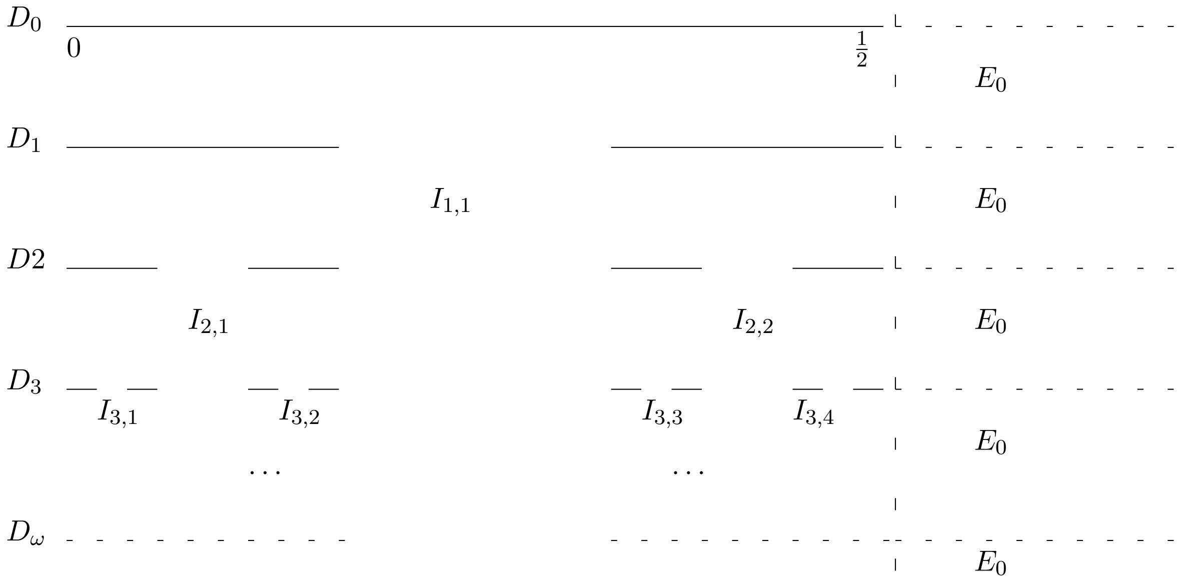
The above theorem tells us that the ordinal α in the equation $\approx\; = \mathbf{O}^{\alpha}(\sim)$ may be employed to "measure" the gap between state bisimilarity and event bisimilarity. In the following, we employ a *Smith-Volterra-Cantor set* (or simply SVC set) to construct an example to show that state bisimilarity can not be obtained by iterating ω times the operator $\mathbf{O}$ from event bisimilaity. This example illustrates the gaps between these two bisimilarities and further between the two approaches for approximating bisimilarity: top-down and bottom-up. Also this example shows that $\mathbf{O}$ is not downward continuous (Corollary 4.7). But, if $\langle S, \mathcal{A}\rangle$ is analytical or discrete, then state bisimilarity and event bisimilarity coincide and the operator $\mathbf{O}$ is continuous ([4] and [9]).

Example 4.2 (SVC-set-construction) We define a sequence of partitions $\Pi_i (i \geq 0)$ and corresponding equivalence relations $R_i (i \geq 0)$ of $S = [0, 1]$ inductively as follows. For an interval I of S, let $\mathcal{B}(I)$ denote the σ-algebra of Borel sets in I, $\mathcal{C}(I)$ the countable subclass that generates the σ-algebra and $\mathcal{M}(I)$ the Lebesgue completion. There is a non-Lebesgue-measurable subset E_0 of $[\frac{1}{2}, 1]$. Let $\mathcal{B}_0$ denote $\sigma(\mathcal{C}[\frac{1}{2}, 1]) \cup \{[0, \frac{1}{2}], E_0\})$. The construction will proceed in steps. At the first step, let $I_{1,1}$ denote the open interval $(\frac{1}{2} \cdot \frac{3}{8}, \frac{1}{2} \cdot \frac{5}{8})$. Thus $I_{1,1}$ is the open middle of the interval $I_0 := [0, \frac{1}{2}]$ of length $\frac{1}{2} \cdot \frac{1}{2^{2 \cdot 1}}$. The second step involves performing the first step on each of the two remaining closed intervals of $I_0 \setminus I_{1,1}$. That is, we produce two open intervals $I_{2,1}$ and $I_{2,2}$, each being the open middle with length $\frac{1}{2} \cdot \frac{1}{2^{2\cdot 2}}$ of one of the two intervals compromising $I_0 \setminus I_{1,1}$. At the i-th step we produce 2^{i-1} open intervals, $I_{i,1}, I_{i,2}, \cdots, I_{i,2^{i-1}}$, each of length $\frac{1}{2} \cdot \frac{1}{2^{2i}}$. The $(i+1)$-th step consists of producing open middles of length $\frac{1}{2} \cdot \frac{1}{2^{2(i+1)}}$ of each of the intervals of

$$I_0 \setminus \bigcup_{j=1}^{i} \bigcup_{k=1}^{2^{j-1}} I_{j,k}.$$

D_0 denotes $[0, \frac{1}{2}]$. For any natural number i, let D_i denote the set $I_0 \setminus \bigcup_{j=1}^{i} \bigcup_{k=1}^{2^{j-1}} I_{j,k}$. With D_ω denoting the SVC set with respect to $[0, \frac{1}{2}]$, we define its complement by

$$I_0 \setminus D_\omega = \bigcup_{j=1}^{\infty} \bigcup_{k=1}^{2^{j-1}} I_{j,k}$$



There are some facts about this SVC set D_ω that we need for the following construction.

(i) D_ω is a closed set and has a positive measure $\frac{1}{4}$. In fact, the total sum of the lengths of the deleted open intervals is $\sum_{i=1}^{\infty} \frac{1}{2} \frac{1}{2^{i+1}} = \frac{1}{4}$.

(ii) D_ω contains a non-Lebesgue-measurable subset $D_{\omega+1}$, since D_ω has a positive measure.

(iii) D_ω is totally disconnected,i.e., all connected components are singletons. That is to say, each connected component in D_ω is a singleton.

Set $R_0 := \{(x, x) : x \in (\frac{1}{2}, 1]\} \cup \{(x, y) : x, y \in [0, \frac{1}{2}]\}$. Next we define another equivalence relation R_1 on $[0, 1]$ which refines R_0 by simulating the trisection process in the construction of the SVC set D_ω.

$$\begin{aligned} R_1 := &\{(x, x) : x \in (\frac{1}{2}, 1]\} \\ &\cup\{(x, y) : x, y \in I_{1,1}\} \\ &\cup\{(x, y) : x, y \in I_0 \setminus I_{1,1}\} \end{aligned}$$

More generally, we define, for $i \geq 1$,

$$\begin{aligned} R_i := &\{(x,x) : x \in (\frac{1}{2}, 1]\} \\ &\cup \bigcup_{j=1}^{i} \{(x,y) : x, y \in \bigcup_{k=1}^{2^{j-1}} I_{j,k}\} \\ &\cup \{(x,y) : x, y \in I_0 \setminus \bigcup_{j=1}^{i} \bigcup_{k=1}^{2^{j-1}} I_{j,k}\} \end{aligned}$$

R_ω denotes the intersection of all R_i's, i.e., $R_\omega = \bigcap_i R_i$. Actually R_ω can be expressed as follows:

$$\begin{aligned} R_\omega = &\{(x,x) : x \in (\frac{1}{2}, 1]\} \\ &\cup \bigcup_{j=1}^{\infty} \{(x,y) : x, y \in \bigcup_{k=1}^{2^{j-1}} I_{j,k}\} \\ &\cup \{(x,y) : x, y \in I_0 \setminus \bigcup_{j=1}^{\infty} \bigcup_{k=1}^{2^{j-1}} I_{j,k}\} \end{aligned}$$

Note that $I_0 \setminus \bigcup_{j=1}^{\infty} \bigcup_{k=1}^{2^{j-1}} I_{j,k}$ is precisely the SVC set D_ω with respect to $[0, \frac{1}{2}]$. Define $\mathcal{B}_n := \sigma(\mathcal{C}[\frac{1}{2}, 1] \cup \{[0, \frac{1}{2}], E_0\} \cup \{D_j : j \preceq n\})$ for $n \preceq \omega + 1$ where $\preceq$ is the ordinal relation.

Let $\mathcal{C}_{\omega+1}$ denote $\mathcal{C}[\frac{1}{2}, 1] \cup \{[0, \frac{1}{2}], E_0\} \cup \{D_i : i = 1, 2, \cdots\} \cup \{D_{\omega+1}\}$, $\mathcal{B}_{-i}$ the σ-algebra $\sigma(\mathcal{C}_{\omega+1} \setminus \{D_i\})(i = 1, 2, \cdots)$ and $\mathcal{B}_{-(\omega+1)}$ the σ-algebra $\sigma(\mathcal{C}_{\omega+1} \setminus \{D_{\omega+1}\})$. Note that all the events in $\mathcal{C}_{\omega+1}$ are Lebesgue-measurable except $D_{\omega+1}$ and E_0. So $\mathcal{B}_{\omega+1} = \sigma(\mathcal{C}_{\omega+1})$ and is countably generated. The following Extension Theorem is the most important "weapon" that we will use to construct our Markov kernel τ.

Proposition 4.3 *(Theorem 1.12.14 in [2]) Assume that*

(i) *μ is a finite nonnegative measure on the measurable space $\langle \Omega, \Sigma \rangle$; and*

(ii) *A is a subset of Ω such that $\mu_*(A) < \mu^*(A)$ where μ_* and μ^* are the inner and outer measures of μ, respectively.*

Then, for any r such that $\mu_*(A) \le r \le \mu^*(A)$, there is a countably additive measure μ' on the σ-algebra $\sigma(\Sigma \cup \{A\})$ such that $\mu'(A) = r$ and $\mu' = \mu$ on Σ.

By appealing to the above theorem, we obtain a measure λ_ω on $\mathcal{B}_\omega$ such that λ_ω is an extension of the Lebesgue measure on the sub-σ-algebra generated by $\mathcal{C} \setminus \{E_0, D_{\omega+1}\}$. It is easy to see that, since $D_{\omega+1}$ is a non-Lebesgue-measurable subset of $[0, \frac{1}{2}]$, $(\lambda_\omega)_*(D_{\omega+1}) < (\lambda_\omega)^*(D_{\omega+1})$. According to the above Extension Theorem, for any r such that $(\lambda_\omega)_*(D_{\omega+1}) \le r \le (\lambda_\omega)^*(D_{\omega+1})$, there is a countably additive extension $\lambda^r_{\omega+1}$ such that $\lambda^r_{\omega+1} = \lambda_\omega$ on $\mathcal{B}_\omega$ and $\lambda^r_{\omega+1}(D_{\omega+1}) = r$. Let $I_{\omega+1} = \{r : (\lambda_\omega)_*(D_{\omega+1}) \le r \le (\lambda_\omega)^*(D_{\omega+1})\}$. There is an injective and *increasing* f from C to the set $I_{\omega+1}$. For each $x \in C$, if $f(x) = r$, then we also use $\lambda^{f(x)}_{\omega+1}$ to denote $\lambda^r_{\omega+1}$. Especially, we simply use $\lambda_{\omega+1}$ to denote the "last" such extension $\lambda^{f(\frac{1}{2})}_{\omega+1}$. Note that $\lambda^{f(x)}_{\omega+1}$ is a measure on $\mathcal{B}_{\omega+1}$ for all $x \in C$.

It is easy to check that, for each $D_i (i = 1, 2, \cdots)$,

$$(\lambda_{\omega+1}\upharpoonright_{\mathcal{B}_{-i}})_*(D_i) = \lambda(D_{i+1}) < \lambda(D_{i-1}) = (\lambda_{\omega+1}\upharpoonright_{\mathcal{B}_{-i}})^*(D_i)$$

Similarly, according to Theorem 4.3, there is a measure λ_{-i} on $\mathcal{B}_{\omega+1}$ such that $\lambda_{-i}(D_i) \neq \lambda_{\omega+1}(D_i)$ and $\lambda_{-i} = \lambda_{\omega+1}$ on $\mathcal{B}_{-i}$.

Now we define a Markov kernel on the measurable space $\langle S, \mathcal{B}_{\omega+1}\rangle$.

$$\tau(x,E) = \begin{cases} x \cdot \lambda_{\omega+1}(E) \text{ if } x \in [\frac{1}{2}, 1], \\ \frac{1}{2} \cdot \lambda_{\omega+1}^{f(x)}(E) \text{ if } x \in C \setminus \{\frac{1}{2}\}, \\ \frac{1}{2} \cdot \lambda_{-i}(E) \text{ if } x \in \bigcup_{k=1}^{2^{i-1}} I_{i,k} \end{cases}$$

Lemma 4.4 *The above defined $M := \langle S, \mathcal{B}_{\omega+1}, \tau\rangle$ is a Markov process.*

Proof. The crucial part is to show that $\tau(\cdot, D_{\omega+1})$ is $\mathcal{B}_{\omega+1}$-measurable. This follows from the fact that f is injective and increasing. □

Lemma 4.5 $[\![\mathcal{L}]\!]_M := \{[\![\phi]\!]_M : \phi \in \mathcal{L}\} \subseteq \{E : E = E_1 \cup [0, \frac{1}{2}]$ *for some* $E_1 \in \mathcal{B}[\frac{1}{2}, 1]\}$. *And the logical equivalence or event bisimilarity $\sim_M$ is*

$$R_0 = \{(x,x) : x \in (\tfrac{1}{2}, 1]\} \cup \{(x,y) : x, y \in [0, \tfrac{1}{2}]\}$$

Theorem 4.6 *(Main Theorem) For simplicity, let $\mathcal{A}$ denote the reference σ-algebra $\mathcal{B}_{\omega+1}$. For the above sequences of σ-algebra $\mathcal{B}_i$ and of equivalence relations R_i, they satisfy the interrelations illustrated as follows:*

$$\begin{tikzcd}
\mathcal{B}_{\omega+1} & \supsetneq & \sigma(\cup_i \mathcal{B}_i) \arrow[d, "\boldsymbol{R}^T(\cdot)"] & \cdots & \supsetneq & \mathcal{B}_1 \arrow[d, "\boldsymbol{R}^T(\cdot)"] & \supsetneq & \mathcal{B}_0 \arrow[d, "\boldsymbol{R}^T(\cdot)"] \arrow[drr, "\boldsymbol{R}(\cdot)"] & \supsetneq & \sigma([\![\mathcal{L}]\!]_M \arrow[d, "\boldsymbol{R}^T(\cdot)"] \\
\cdots & \subseteq & R_\omega \arrow[ull, "\mathcal{A}_{(\cdot)}"] & \cdots & \subsetneq & R_2 \arrow[ull, "\mathcal{A}_{(\cdot)}"] & \subsetneq & R_1 \arrow[ull, "\mathcal{A}_{(\cdot)}"] & \subsetneq & R_0(=\sim_M) \arrow[ull, "\mathcal{A}_{(\cdot)}"]
\end{tikzcd}$$

Corollary 4.7 $\mathcal{A}(R_\omega) = \mathcal{B}_{\omega+1}$ *and* $\boldsymbol{R}^T(\mathcal{B}_{\omega+1}) \subsetneq R_\omega$. *So* $\bigcap_i \mathbf{O}(R_i) \supsetneq \mathbf{O}(\bigcap_i R_i)$ *and hence* $\mathbf{O}$ *is not downward continuous.*

Proof. The first part is straightforward. The second one follows from the fact

$$\mathbf{R}^T(\mathcal{B}_{\omega+1}) = \{(x,x) : x \in [\tfrac{1}{2}, 1] \cup C\} \cup \bigcup_{j=1}^{\infty} \bigcup_{k=1}^{2^{j-1}} I_{j,k}$$

and hence $\mathbf{R}^T(\mathcal{B}_{\omega+1}) \subsetneq R_\omega$. □

5 Filtration and Hennessy-Milner property

In this section, we simulate Goldblatt's work in [10] to develop a general theory about the relationship among bisimilarity, filtration and Hennessy-Milner property by providing another characterization of the Hennessy-Milner property through filtration (Theorem 5.4). The following proposition from [4] tells us that event bisimilarity is characterized by the simple logic $\mathcal{L}$.

Theorem 5.1 *For the Markov process $M = \langle S, \mathcal{A}, \tau\rangle$,*

(i) *$\langle S, \sigma([\![\mathcal{L}]\!]_M), \tau\rangle$ is a Markov process;*

(ii) *$\boldsymbol{R}(\mathcal{A}(\sim)) = \sim$;*

(iii) *$\sigma([\![\mathcal{L}]\!]_M)$ is the smallest stable sub-σ-algebra $\mathcal{A}'$ that defines $\sim$, i.e., $\boldsymbol{R}(\mathcal{A}') = \sim$.*

From the above proposition, we know that $\mathcal{A}(\sim)$ is the biggest σ-algebra that defines $\sim$ but is generally not the biggest *stable* σ-algebra that defines $\sim$ because otherwise $\sim = \approx$ (Theorem 5.4).

However, the biggest stable σ-algebra that defines $\sim$ always exists. Let $\mathbf{F} = \{\mathcal{B} : \mathcal{A}(\sim) \supseteq \mathcal{B} \supseteq \sigma([\![\mathcal{L}]\!]_M), \langle S, \mathcal{B}, \tau\rangle \text{ is a Markov process }\}$. $\mathbf{F}$ is a complete lattice under the following lattice operations: for $(\mathcal{B}_i)_{i\in I} \subseteq \mathbf{F}$,

- $\bigwedge_i \mathcal{B}_i = \bigcap_i \mathcal{B}_i$;
- $\bigvee_i \mathcal{B}_i = \bigcap\{\mathcal{B} \in \mathbf{F} : \mathcal{B} \supseteq \mathcal{B}_i \text{ for all } i \in I\}$.

Let $\mathcal{F}$ denote $\bigcup \mathbf{F}$. It follows immediately that $\langle S, \mathcal{F}, \tau\rangle$ is a Markov process and is the biggest stable sub-σ-algebra that is contained in $\mathcal{A}(\sim)$ and defines event bisimilarity $\sim$.

In the following, we give a general definition of filtration. Essentially, a filtration of a Markov process M' through a sublanguage $\mathcal{L}'$ is its quotient that respects the satisfiability of all formulas in $\mathcal{L}'$. Let $\mathcal{L}'$ be a subset of language of $\mathcal{L}$ which is *closed under subformulas.* In other words,

- $\top \in \mathcal{L}'$;
- if $L_r\phi \in \mathcal{L}'$, $\phi \in \mathcal{L}'$;
- if $\phi \wedge \psi \in \mathcal{L}'$, $\phi \in \mathcal{L}'$ and $\psi \in \mathcal{L}'$.

$\mathcal{L}'$ defines an equivalence relation $\sim_{\mathcal{L}'}$ on S: $s \sim_{\mathcal{L}'} t$ if they satisfy the same set of formulas in $\mathcal{L}'$. Any Markov processes $M_{\sim_{\mathcal{L}'}} = \langle S/_{\sim_{\mathcal{L}'}}, \mathcal{A}', \tau^{\mathcal{A}'}_{\sim_{\mathcal{L}'}}\rangle$ on the set $S/_{\sim_{\mathcal{L}'}}$ of equivalence classes where $\mathcal{A}' \subseteq \mathcal{A}/_{\sim_{\mathcal{L}'}}$ is called a *filtration of M through the sub-language* $\mathcal{L}'$ if it satisfies the following property: for any $s \in S$ and $\phi \in \mathcal{L}'$,

$$M, s \models \phi \text{ if and only if } M_{\sim_{\mathcal{L}'}}, [s]_{\mathcal{L}'} \models \phi$$

When the context is clear, we simply call $M_{\sim_{\mathcal{L}'}}$ a filtration. For the measurable space $\langle S/_{\sim_{\mathcal{L}'}}, \sigma([\![\mathcal{L}]\!]_M)/_{\sim_{\mathcal{L}'}}\rangle$, let $\tau^{\sigma([\![\mathcal{L}]\!]_M)/_{\sim_{\mathcal{L}'}}}_{\sim_{\mathcal{L}'}}([s]_{\mathcal{L}'}, E) := \tau(s', \bigcup E)$ for *some* $s' \in [s]_{\mathcal{L}'}$ and $E \in \sigma([\![\mathcal{L}]\!]_M)/_{\sim_{\mathcal{L}'}}$. From Proposition 2.2, we know $\sigma([\![\mathcal{L}]\!]_M)/_{\sim_{\mathcal{L}'}}$ is stable.

Theorem 5.2 *$\langle S/_{\sim_{\mathcal{L}'}}, \sigma([\![\mathcal{L}]\!]_M)/_{\sim_{\mathcal{L}'}}, \tau^{\sigma([\![\mathcal{L}]\!]_M)/_{\sim_{\mathcal{L}'}}}_{\sim_{\mathcal{L}'}}\rangle$ is a fitration.*

It is clear that, for any filtration $M_{\sim_{\mathcal{L}'}} = \langle S/_{\sim_{\mathcal{L}'}}, \mathcal{A}', \tau_{\sim_{\mathcal{L}'}}\rangle$, $\mathcal{A}' \supseteq \sigma([\![\mathcal{L}']\!]_{M_{\sim_{\mathcal{L}'}}})$ and $\langle S/_{\sim_{\mathcal{L}'}}, \sigma([\![\mathcal{L}']\!]_{M_{\sim_{\mathcal{L}'}}}), \tau_{\sim_{\mathcal{L}'}}\rangle$ is a filtration.

In the following, we employ the idea of averaging in [3] to show that, for any σ-algebra $\mathcal{A}'$ such that $\mathcal{A}/_{\sim_{\mathcal{L}'}} \supseteq \mathcal{A}' \supseteq \sigma([\![\mathcal{L}']\!]_{M_{\sim_{\mathcal{L}'}}})$, there is always a filtration $\langle S/_{\sim_{\mathcal{L}'}}, \mathcal{A}', \tau_{\sim_{\mathcal{L}'}}\rangle$ with $\mathcal{A}'$ as its σ-algebra of events. The main task is to find a Markov kernel $\tau_{\sim_{\mathcal{L}'}}$ such that $\langle S/_{\sim_{\mathcal{L}'}}, \mathcal{A}', \tau_{\sim_{\mathcal{L}'}}\rangle$ is a Markov process.

In order to apply averaging here, we assume that there is a prior probability measure P on the measure space $\langle S, \mathcal{A}\rangle$. Note that any $\sim_{\mathcal{L}'}$-equivalence class $[s]_{\sim\mathcal{L}'}$ is $\mathcal{A}$-measurable. We define a mapping $\tau^{\mathcal{A}'}_{\sim_{\mathcal{L}'}} : S_{\sim'_{\mathcal{L}}} \times \mathcal{A}' \to [0,1]$ as follows: for any $[s]_{\mathcal{L}'} \in S_{\sim'_{\mathcal{L}}}$ and $A' \in \mathcal{A}'$,

$$\tau^{\mathcal{A}'}_{\sim_{\mathcal{L}'}}([s]_{\mathcal{L}'}, A') := \frac{\displaystyle\int_{[s]_{\mathcal{L}'}} \tau(s, \bigcup A')dP(s)}{P([s]_{\mathcal{L}'})}.$$

From [3] and Proposition 2.2, we know that

Theorem 5.3 *For such defined $\tau_{\sim_{\mathcal{L}'}}^{\mathcal{A}'}$,*

(i) $\langle S/_{\sim_{\mathcal{L}'}}, \mathcal{A}', \tau_{\sim_{\mathcal{L}'}}^{\mathcal{A}'}\rangle$ *is a Markov process and hence is a filtration of M through the sub-language $\mathcal{L}'$.*

(ii) $\langle S, \mathcal{A}'_{\cup}, \tau\rangle$ *is a Markov process if and only if the natural mapping from M to $\langle S/_{\sim_{\mathcal{L}}}, \mathcal{A}', \tau_{\sim}^{\mathcal{A}_\sim}\rangle$ is* a zigzag morphism, *i.e., for any $s \in S$ and $A' \in \mathcal{A}'$, $\tau_{\sim_{\mathcal{L}'}}^{\mathcal{A}'}([s]_{\sim_{\mathcal{L}'}}, A') = \tau(s, \bigcup A')$.*

For the language $\mathcal{L}'$, $\langle S/_{\sim_{\mathcal{L}'}}, \sigma([\![\mathcal{L}]\!]_M)/_{\sim_{\mathcal{L}'}}, \tau_{\sim_{\mathcal{L}'}}^{\sigma([\![\mathcal{L}']\!]_M)/\sim_{\mathcal{L}'}}\rangle$ is called the *smallest filtration* and $\langle S/_{\sim_{\mathcal{L}'}}, \mathcal{A}/_{\sim_{\mathcal{L}'}}, \tau_{\sim_{\mathcal{L}'}}^{\mathcal{A}/\sim_{\mathcal{L}'}}\rangle$ the *greatest filtration* on $S/_{\sim_{\mathcal{L}'}}$. Note that generally the natural mapping from M to $\langle S/_{\sim_{\mathcal{L}'}}, \mathcal{A}', \tau_{\sim_{\mathcal{L}'}}^{\mathcal{A}'}\rangle$ is not a zigzag morphim because $\tau_{\sim_{\mathcal{L}'}}^{\mathcal{A}'}$ may be different from τ. That is to say, generally we don't have $\tau_{\sim_{\mathcal{L}'}}^{\mathcal{A}'}([s]_{\sim_{\mathcal{L}'}}, A') = \tau(s, \bigcup A')$ for $s \in S$ and $A' \in \mathcal{A}'$. But, for $A' \in \sigma([\![\mathcal{L}]\!]_{M_{\mathcal{L}}})$, we always have that, for any $s \in S$, $\tau_{\sim_{\mathcal{L}'}}^{\mathcal{A}'}([s]_{\sim_{\mathcal{L}'}}, A') = \tau(s, \bigcup A')$.

In [15], we provides a sequence of filtrations through a sequence of *finite languages* $(\mathcal{L}_i)_{i=1}^{\infty}$, which are closed under subformulas, to approximate the original Markov process M. This is a kind of approximation based on the so-called *bottom-up* approximating (*event*) bisimilarity.

The following theorem is a generalization of Theorem 15 in [10] and provides another characterization of the Hennessy-Milner property through filtration.

Theorem 5.4 *For the whole language $\mathcal{L}$,*

(i) *the natural mapping from M to $\langle S/_{\sim_{\mathcal{L}}}, \mathcal{A}/_{\sim}, \tau_{\sim}^{\mathcal{A}(\sim)}\rangle$ is a zigzag morphism if and only if $\mathcal{A}(\approx) = \mathcal{A}(\sim)$ or equivalently $\approx\, = \,\sim$, i.e., M satisfies the Hennessy-Milner property.*

(ii) $\mathcal{A}(\sim) = \sigma([\![\mathcal{L}]\!]_M)$ *iff there is only one filtration through the language $\mathcal{L}$ iff $\approx\, = \,\sim$.*

Proof. For the second part, we note that there is only one filtration through the language $\mathcal{L}$ iff, for each $A' \in \mathcal{A}(\sim)$, $\tau(\cdot, A')$ is constant on $[s]_{\sim_{\mathcal{L}}}$ for every $s \in S$. □

The following is about the position of filtration in a general picture of interrelationships among different σ-algebras and equivalence relations.

$$\begin{tikzcd}
 & filtration & & \rightarrow & \mathcal{A}/_{\sim} & \supseteq & \mathcal{F}/_{\sim} & \supseteq & \sigma([\![\mathcal{L}]\!])/_{\sim} \\
A \arrow[d, leftrightarrow, "\mathcal{A}_{(\cdot)}"', "R_{(\cdot)}"] & \supseteq & \mathcal{A}_{\approx} \arrow[d, leftrightarrow, "\mathcal{A}_{(\cdot)}"', "R_{(\cdot)}"] & \supseteq & \mathcal{A}_{\sim} \arrow[u, "(\cdot)/_{\sim}"] \arrow[ddr, phantom] & \supseteq & \mathcal{F} \arrow[u, "(\cdot)/_{\sim}"] \arrow[d, "R_{(\cdot)}"] & \supseteq & \sigma([\![\mathcal{L}]\!]) \arrow[u, "(\cdot)/_{\sim}"] \arrow[dll, "R_{(\cdot)}"] \\
R_{\mathcal{A}} & \subseteq & \approx & & \subseteq & & \sim \arrow[ull, leftrightarrow, "\mathcal{A}_{(\cdot)}", "R_{(\cdot)}"'] & &
\end{tikzcd}$$

6 Conclusion

In this paper, we study the difference between event and state bisimilarities from the perspective of fixed point theory. We quantify this difference by counting the iteration times of the operator **O** from event bisimilarity to state bisimilarity. Our work provides insights about the Hennessy-Milner property for general Markov processes. At the end of this paper, we provide another characterization of this property through filtration. Approximate bisimilarity [14] is another important notion to reason about approximate equivalence of processes. It is a subject for future work to study approximating bisimilarity for Markov processes from the perspective of approximate bisimilarity.

References

[1] P. Billingsley. *Measure and Probability.* John Wiley & Sons, Inc., third edition, 1995. Wiley Series in Probability and Mathematical Statistics.

[2] V.I. Bogachev. *Measure Theory.* Springer-Verlag Berlin Heidelberg, 2007.

[3] P. Chaput, V. Danos, P . Panangaden, and G. D. Plotkin. Approximating markov processes by averaging. In *ICALP (2)*, pages 127–138, 2009.

[4] V. Danos, Josee Desharnais, François Laviolette, and Prakash Panangaden. Bisimulation and cocongruence for probabilistic systems. *Inf. Comput.*, 204(4):503–523, 2006.

[5] J Desharnais, A. Edalat, and Panangaden P. Bisimulation for labeled markov processes. *Inf. and Compt.*, 179:163–193, 2002.

[6] J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Approximating labeled markov processes. *Inf. and Comp.*, 184:160–200, 2003.

[7] E. E Doberkat. Stochastic relations: congruence, bisimulations and the hennessy-milner theorem. *SIAM J. Computing*, 35(3):590–626, 2006.

[8] E. E Doberkat. *Stochastic Coalgebraic Logic.* EATCS Mongraphs in Theoretical Computer Sciences. Springer, 2010.

[9] E. E Doberkat. Lattice properties of congruences for stochastic relations. *Ann. Pure Appl. Logic*, 163(8):1016–1029, 2012.

[10] R. Goldblatt. Saturation and the hennessy-milner property. In M. de Rijke A. Ponse and Y. Venema, editors, *Modal Logic and Process Algebra*, pages 107–129. CSLI Publications, Stanford, CA, 1995.

[11] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic.* Foundation of Computing Series. MIT Press, Cambridge, 2000.

[12] P. Terraf. Unprovability of the logical characterization of bisimulation. *Inf. Comput.*, 209(7):1048–1056, 2011.

[13] F. van Breugel and Worrell J. Approximating and computing behavioural distances in probabilistic transition systems. *Theor. Comput. Sci.*, 360(1-3):373–385, 2006.

[14] M. Ying and M. Wirsing. Approximate bisimilarity. In *AMAST*, pages 309–322, 2000.

[15] C. Zhou and M. Ying. Approximation of markov processes through filtration. *Theoretical Comput. Sci.*, 446:75–97, 2012.