Electronic Notes in Theoretical Computer Science

Mathematical Foundations of Programming Semantics
Thirtieth Annual Conference

Cornell University
Ithaca, NY
June 12–15, 2014

Preliminary Proceedings

Guest Editors:
Bart Jacobs, Alexandra Silva and Sam Staton
## Contents

Preface ................................................................. iv

**Jiří Adámek, Robert S. R. Myers, Henning Urbat**  
*On Continuous Nondeterminism and State Minimality*  .................................................... 1

**Danel Ahman and Tarmo Uustalu**  
*Coalgebraic Update Lenses*  ........................................... 22

**Dariusz Biernacki and Sergueï Lenglet**  
*Applicative Bisimilarities for Call-by-Name and Call-by-Value λµ-Calculus*  ................................... 44

**Stephen Brookes**  
*On Grainless Footprint Semantics for Shared-memory Programs*  ........................................ 73

**Simon Castellan, Jonathan Hayman, Marc Lasson and Glynn Winskel**  
*Strategies as Concurrent Processes*  ........................................ 93

**Liang-Ting Chen and Achim Jung**  
*On a Categorical Framework for Coalgebraic Modal Logic*  .................................................. 112

**J.R.B. Cockett, P.J.W. Hofstra and P. Hrubeš**  
*Total Maps of Turing Categories*  ........................................ 132

**Philippa Gardner, Azalea Raad, Mark Wheelhouse and Adam Wright**  
*Abstract Local Reasoning for Concurrent Libraries: Mind the Gap*  ...................................... 150

**Jean Goubault-Larrecq and Achim Jung**  
*QRB, QFS, and the Probabilistic Powerdomain*  .................................................. 170

**Samy Jaziri, Kim G. Larsen, Radu Mardare and Bingtian Xue**  
*Adequacy and Complete Axiomatization for Timed Modal Logic*  ........................................ 186

**Dexter Kozen, Radu Mardare and Prakash Panangaden**  
*A Metrized Duality Theorem for Markov Processes*  .................................................. 215


Marc Lasson  
Canonicity of Weak ω-Groupoid Laws Using Parametricity Theory  232

Giulio Manzonetto and Domenico Ruoppolo  
Relational Graph Models, Taylor Expansion and Extensionality  247

Maciej Pirog and Jeremy Gibbons  
The Coinductive Resumption Monad  273

Mathys Rennela  
Towards a Quantum Domain Theory: Order-enrichment and Fixpoints in W*-algebras  289

Tetsuya Sato  
Identifying All Preorders on the Subdistribution Monad  308
Preface

This volume collects papers presented at the 30th Annual Conference on Mathematical Foundations of Programming Semantics (MFPS XXX), held on the campus of Cornell University, Ithaca, New York, USA, from Thursday, June 12 through Sunday, June 15, 2014. 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 XXX were overseen by Dexter Kozen.

The MFPS XXX Program Committee members are:

Bart Jacobs, Alexandra Silva, Sam Staton (Co-chairs), Radboud University Nijmegen

- Steve Awodey, CMU
- Andrej Bauer, Ljubljana
- Stephen Brookes, CMU
- Pierre Clairambault, ENS Lyon
- Martín Escardó, Birmingham
- Fabio Gadducci, Pisa
- Ichiro Hasuo, Tokyo
- Martin Hofmann, LMU München
- Achim Jung, Birmingham
- Shin-ya Katsumata, Kyoto
- Naoki Kobayashi, Tokyo
- Dexter Kozen, Cornell
The Program Committee selected 16 out of 32 submitted papers for presentation at the meeting. The Organizing Committee selected the following people to give invited plenary talks at the meeting:

- Nick Benton, Microsoft Research, Cambridge, UK
- Andy Gordon, Microsoft Research, Cambridge, UK
- Elham Kashefi, University of Edinburgh, UK
- Dexter Kozen, Cornell University, USA
- Prakash Panangaden, McGill University, Canada
- Alexandra Silva, Radboud University, Nijmegen, The Netherlands

In addition, two of these invited speakers organised special sessions:

- Probabilistic computing, organized by Andy Gordon
- Quantum computing, organized by Elham Kashefi

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 would also like to thank the US Office of Naval Research for its continued support of the MFPS series. Finally, we would like to thank Cornell University for providing the venue and the local staff who helped with the preparations and execution of the conference, especially Michelle Eighmey and Molly Trufant.

Bart Jacobs
Alexandra Silva
Sam Staton
On Continuous Nondeterminism and State Minimality

Jiří Adámek, Robert S. R. Myers, Henning Urbat

Institut für Theoretische Informatik
Technische Universität Braunschweig
Germany

Stefan Milius
Lehrstuhl für Theoretische Informatik
Friedrich-Alexander-Universität Erlangen-Nürnberg
Germany

Abstract
This paper is devoted to the study of nondeterministic closure automata, that is, nondeterministic finite automata (nfas) equipped with a strict closure operator on the set of states and continuous transition structure. We prove that for each regular language \( L \) there is a unique minimal nondeterministic closure automaton whose underlying nfa accepts \( L \). Here minimality means no proper sub or quotient automata exist, just as it does in the case of minimal dfas. Moreover, in the important case where the closure operator of this machine is topological, its underlying nfa is shown to be state-minimal. The basis of these results is an equivalence between the categories of finite semilattices and finite strict closure spaces.

Keywords: Canonical Nondeterministic Automata, State Minimality, Closure Spaces, Semilattices

1 Introduction

Why are state-minimal deterministic finite automata (dfas) easy to construct, whilst no efficient minimization procedure for nondeterministic finite automata (nfas) is known? Let us start with the observation that minimal dfas are built inside the category \( \text{Set}_f \) of finite sets and functions and are characterized by having no proper subautomata (reachability) and no proper quotient automata (simplicity). Nfas can be regarded as dfas interpreted in the category \( \text{Rel}_f \) of finite sets and relations, and so one might hope to build minimal nfas in the same way as minimal dfas, but now in \( \text{Rel}_f \). However, there is a significant difference: \( \text{Set}_f \) is both finitely complete and cocomplete, yet \( \text{Rel}_f \) does not have coequalizers, i.e., canonical quotients. The lack of such canonical constructions provides evidence for the lack of canonical state-minimal nfas.
This suggests the following strategy: form the cocompletion of $\text{Rel}_f$ obtained by freely adding canonical quotients, which turns out to be the category $\text{JSL}_f$ of finite join-semilattices (see Appendix), and build minimal automata in this larger category. Every nfa may be viewed as a dfa in $\text{JSL}_f$ via the usual subset construction. In order to obtain more efficient presentations of nfas, avoiding the full power set of states, we make use of a categorical equivalence between $\text{JSL}_f$ and the category $\text{Cl}_f$ of finite strict closure spaces [2]. The objects of the latter are finite sets $Z$ equipped with a strict closure operator (i.e., an extensive, monotone and idempotent map $\text{cl}_Z : \mathcal{P}Z \to \mathcal{P}Z$ preserving the empty set), and the morphisms are continuous relations, see Definitions 2.9 and 2.10 below. For example, every finite topological space induces a finite strict closure space; these closures are called topological.

Just as nfas may be viewed as deterministic automata interpreted in $\text{Rel}_f$, nondeterministic closure automata (ncas) are deterministic automata interpreted in $\text{Cl}_f$: an nca is an nfa with a strict closure operator on its set of states, continuous transition relations, an open set of final states and a closed set of initial states. Since the category $\text{Cl}_f$ has the same relevant properties as $\text{Set}_f$, we derive for each regular language $L \subseteq \Sigma^*$ the existence of a unique minimal nca $\mathcal{N}(L)$ whose underlying nfa (forgetting the closure operator) accepts $L$. It is minimal in the sense that it has no proper subautomata (reachability) and no proper quotient automata (simplicity), and can be constructed in a way very much analogous to Brzozowski’s classical construction of the minimal dfa [6]: starting with any nca $\mathcal{N}$ accepting $L$, one has

$$\mathcal{N}(L) = \text{reach} \circ \text{rev} \circ \text{reach} \circ \text{rev}(\mathcal{N})$$

where $\text{reach}$ and $\text{rev}$ are continuous versions of the reachable subset construction and the reversal operation for nfas, respectively.

The states of $\mathcal{N}(L)$ are the prime derivatives of $L$, i.e., those non-empty derivatives $w^{-1}L = \{v \in \Sigma^* : vw \in L\}$ of $L$ that do not arise as a union of other derivatives. The underlying nfa of $\mathcal{N}(L)$ accepts $L$, thus it is natural to ask when this nfa is state-minimal. Our main result is:

If the closure of $\mathcal{N}(L)$ is topological then the underlying nfa is state-minimal.

In other words, we identify a natural class of regular languages for which canonical state-minimal nondeterministic acceptors exist.

**Related Work.** Our paper is inspired by the work of Denis, Lemay and Terlutte [7] who define a canonical nondeterministic acceptor for each regular language $L$. In fact, the underlying nfa of our minimal nca $\mathcal{N}(L)$ is precisely their ‘canonical residual finite state automaton’, and our Brzozowski construction of $\mathcal{N}(L)$ in Section 3 generalizes their construction in [7, Theorem 5.2]. The main conceptual difference is that the latter works on the level of nfas, while our construction takes the continuous structure of nondeterministic closure automata into account. We hope to convince the reader that ncas provide the proper setting in which to study these canonical nfas and their construction.

We have introduced nondeterministic closure automata in [2] where we demon-
Adámek, Milius, Myers, Urbat

strated that ncas – as well as related machines like the átomaton of Brzozowski and Tamm [5] – are instances of a uniform coalgebraic construction. There we also gave various simple criteria for nondeterministic state minimality. The present paper can be understood as an in-depth study of ncas, extending the results from [2] and [7] in two ways: firstly, we provide a richer and more conceptual way of constructing the minimal nca $N(L)$ (compared to [7]) by working with closures. Secondly, we prove that the underlying nfa of $N(L)$ is state-minimal provided that $N(L)$ has topological closure, thereby generalizing a much weaker criterion from [2].

2 From Deterministic JSL-Automata to Nondeterministic Closure Automata

In this section we consider deterministic automata interpreted in the category of join-semilattices, and explain how they induce nondeterministic closure automata. We shall assume familiarity with basic concepts from category theory (categories, functors, duality and equivalence).

Notation 2.1 Throughout this paper we fix an alphabet $\Sigma$. The composition of relations $R \subseteq A \times B$ and $S \subseteq B \times C$ is $S \circ R = \{(a, c) : \exists b \in B, (a, b) \in R \land (b, c) \in S\}$. Moreover $R[A'] = \{b \in B : \exists a \in A', (a, b) \in R\}$ denotes the $R$-image of a subset $A' \subseteq A$, and we write $R[a]$ instead of $R[\{a\}]$.

Let us first recall deterministic and nondeterministic finite automata and provide them with suitable morphisms.

Definition 2.2 (1) A nondeterministic finite automaton (nfa) $N = (Z, R_a, F)$ consists of a finite set $Z$ of states, transition relations $R_a \subseteq Z \times Z$ for every $a \in \Sigma$, and a set $F \subseteq Z$ of final states. A pointed nfa $(N, I)$ is additionally equipped with a set of initial states $I \subseteq Z$.

(2) Nfas form a category $\text{Nfa}$ whose morphisms $B : (Z, R_a, F) \to (Z', R'_a, F')$ are relations $B \subseteq Z \times Z'$ that preserve and reflect transitions (i.e. $R'_a \circ B = B \circ R_a$) and moreover $z \in F$ iff $B[z] \cap F' \neq \emptyset$. Likewise we have the category $\text{Nfa}_*$ of pointed nfas, whose morphisms $B : (N, I) \to (N', I')$ are additionally required to satisfy $B[I] = I'$.

Remark 2.3 Our choice of $\text{Nfa}$-morphisms $B$ is sound: for each $z \in Z$ the pointed nfas $(Z, R_a, F, \{z\})$ and $(Z', R'_a, F', B[z])$ accept the same language.

Definition 2.4 (1) A deterministic finite automaton (dfa) is an nfa $D = (Q, \delta_a, F)$ whose transition relations $\delta_a : Q \to Q$ are functions. A pointed dfa $(D, q_0)$ is a dfa equipped with a single initial state $q_0 \in Q$. Morphisms of (pointed) dfas are precisely the $\text{Nfa}$- (resp. $\text{Nfa}_*$)-morphisms that are functions.

(2) A deterministic automaton (da) is defined in analogy to to (1), except that the set of states is not required to be finite.

We are mainly interested in dfas as carrying a semilattice structure.
**Notation 2.5** Let $\text{JSL}$ denote the category of (join-)semilattices with a bottom element $\bot$, whose morphisms are $\bot$-preserving semilattice homomorphisms. $\text{JSL}_f$ is the full subcategory of finite semilattices.

One can view the final states of a da $(Q, \delta_a, F)$ as a predicate $f : Q \rightarrow \{0, 1\}$ with $F = f^{-1}(\{1\})$. This suggests the following definition: a $\text{JSL}$-da is a da whose state set $Q$ carries a semilattice structure, such that the transitions $\delta_a : Q \rightarrow Q$ and the final state predicate $f : Q \rightarrow 2$ are semilattice morphisms. Here 2 denotes the 2-chain $0 < 1$. Note that to give a morphism $f : Q \rightarrow 2$ means precisely to give a prime filter $F \subseteq Q$, i.e., $\bot_Q \notin F$ and $q \lor q' \in F$ iff $q \in F$ or $q' \in F$. Indeed, given $f$, the set $F = f^{-1}(\{1\})$ is a prime filter, and conversely every prime filter of $Q$ arises in this way. Therefore:

**Definition 2.6** (a) A $\text{JSL}$-da is a triple $\mathcal{D} = (Q, \delta_a, F)$ where $Q$ is a semilattice of states, the $a$-transitions $\delta_a : Q \rightarrow Q$ are semilattice homomorphisms for $a \in \Sigma$, and the final states $F \subseteq Q$ form a prime filter. Given another $\text{JSL}$-da $\mathcal{D}' = (Q', \delta'_a, F')$, a morphism $h : \mathcal{D} \rightarrow \mathcal{D}'$ is a semilattice homomorphism $h : Q \rightarrow Q'$ such that $\delta'_a \circ h = h \circ \delta_a$ and $q \in F$ iff $h(q) \in F'$. We denote by $\text{Jda}$ the category of all $\text{JSL}$-das, and by $\text{Jdfa}$ the full subcategory of $\text{JSL}$-dfas, that is, $\text{JSL}$-das with a finite set of states.

(b) A pointed $\text{JSL}$-da $(\mathcal{D}, q_0)$ is a $\text{JSL}$-da $\mathcal{D} = (Q, \delta_a, F)$ with an initial state $q_0 \in Q$. Morphisms of pointed $\text{JSL}$-das must additionally preserve the initial state. $\text{Jda}_*$ denotes the category of pointed $\text{JSL}$-das, and $\text{Jdfa}_*$ the full subcategory of pointed $\text{JSL}$-dfas.

(c) The language accepted by a pointed $\text{JSL}$-da $(\mathcal{D}, q_0)$ is the language accepted by its underlying da, that is, the set

$$\mathcal{L}_{\mathcal{D}}(q_0) = \{w \in \Sigma^* : \delta_w(q_0) \in F\}$$

where $\delta_w : Q \rightarrow Q$ is the usual inductive extension of the transition function given by $\delta_e = \text{id}_Q$ and $\delta_wa = \delta_a \circ \delta_w$.

**Example 2.7** (1) Take any nfa $N = (Z, R_a, F)$. The usual determinization via the subset construction is a $\text{JSL}$-dfa. Indeed, the states $\mathcal{P}Z$ form a semilattice w.r.t. union, the transitions preserve binary unions and the empty set, and the final states $\{A \subseteq Z : A \cap F \neq \emptyset\}$ form a prime filter.

(2) Let $\mathcal{P}\Sigma^*$ be the semilattice (w.r.t. union) of all languages over $\Sigma$. It carries the structure of a $\text{JSL}$-da whose transitions are given by $L \rightarrow a^{-1}L$ ($a \in \Sigma$) and whose final states $F$ are precisely the languages containing the empty word. This automaton $\mathcal{D}_{\mathcal{P}\Sigma^*} = (\mathcal{P}\Sigma^*, a^{-1}(\cdot), F)$ is easily seen to be the final $\text{JSL}$-da: every $\text{JSL}$-da has a unique $\text{Jda}$-morphism into $\mathcal{D}_{\mathcal{P}\Sigma^*}$, namely the function mapping each state to the language it accepts.

(3) Let $\mathcal{D}_{\text{Reg}(\Sigma)}$ be the subautomaton of $\mathcal{D}_{\mathcal{P}\Sigma^*}$ whose states are the regular languages over $\Sigma$. It can be characterized up to isomorphism by the property that every $\text{JSL}$-dfa has a unique $\text{Jda}$-morphism into $\mathcal{D}_{\text{Reg}(\Sigma)}$. 

![Image](image-url)
Remark 2.8 Readers familiar with the theory of coalgebras will immediately notice that JSL-das correspond to coalgebras for the functor \( T = 2 \times \text{Id}^{\Sigma} : \text{JSL} \to \text{JSL} \). The examples (2) and (3) above then state precisely that \( D_{\Sigma^{1}} \) is the final \( T \)-coalgebra and \( D_{\text{Reg}(\Sigma)} \) is the final locally finite \( T \)-coalgebra, see [9].

In [2] it was proved that the category \( \text{JSL}_{f} \) of finite semilattices is equivalent to the category of finite strict closure spaces. We recall the necessary concepts.

Definition 2.9 A closure operator (shortly, a closure) on a set \( Z \) is a monotone, idempotent and extensive function \( \text{cl}_{Z} : \mathcal{P}Z \to \mathcal{P}Z \), that is,

\[
A \subseteq B \quad \Rightarrow \quad \text{cl}_{Z}(A) \supseteq A, \quad \text{cl}_{Z} \circ \text{cl}_{Z} = \text{cl}_{Z}, \quad \text{for all } A, B \subseteq Z.
\]

A closure space \((Z, \text{cl}_{Z})\) is a set with a closure defined on it. It is finite if \( Z \) is finite and strict if \( \text{cl}_{Z}(\emptyset) = \emptyset \). A subset \( A \subseteq Z \) is closed if \( \text{cl}_{Z}(A) = A \) and open if its complement \( \bar{A} \subseteq Z \) is closed. We write \( \text{cl}_{Z}(z) \) for \( \text{cl}_{Z}(\{z\}) \).

Definition 2.10 Let \( Z_{1} \) and \( Z_{2} \) be finite strict closure spaces. Then a relation \( B \subseteq Z_{1} \times Z_{2} \) is said to be continuous if:

(i) For each \( z \in Z_{1} \), the image \( B[z] \subseteq Z_{2} \) is closed in \( Z_{2} \).

(ii) \( B[\text{cl}_{Z_{1}}(A)] \subseteq \text{cl}_{Z_{2}}(B[A]) \) for all subsets \( A \subseteq Z_{1} \).

Given two continuous relations \( B_{1} : Z_{1} \to Z_{2} \) and \( B_{2} : Z_{2} \to Z_{3} \) we define their continuous composition as follows:

\[
B_{2} \circ B_{1} = \{(z_{1}, z_{3}) \in Z_{1} \times Z_{3} : z_{3} \in \text{cl}_{Z_{3}}(B_{2}(B_{1}[z_{1}]))\} : Z_{1} \to Z_{3}.
\]

That is, one forms the usual relational composition and takes the closure in \( Z_{3} \). The continuous identity is defined by \( \text{id}_{Z} = \{(z, z') \in Z \times Z : z' \in \text{cl}_{Z}(z)\} \).

Definition 2.11 Let \( \text{Cl}_{f} \) denote the category of finite strict closure spaces and continuous relations, with continuous composition and identities.

Remark 2.12 Strict closure spaces can be regarded as generalized topological spaces. Indeed, every topological space \( Z \) induces a strict closure space \((Z, \text{cl}_{Z})\) where \( \text{cl}_{Z} \) is the usual topological closure operator. It preserves finite unions, i.e.,

\[
\text{cl}_{Z}(A \cup B) = \text{cl}_{Z}(A) \cup \text{cl}_{Z}(B) \quad \text{for all } A, B \subseteq Z.
\]

Conversely, every strict closure space \( Z \) satisfying (*) arises from a topological space. Moreover, if \( B : Z_{1} \to Z_{2} \) is a function between topological spaces and \( Z_{2} \) is a \( T_{1} \) space, then \( B \) is continuous in the sense of topology iff it is continuous in the sense of Definition 2.10.

Definition 2.13 A closure \( \text{cl}_{Z} \) satisfying (*) is called topological.

Definition 2.14 Let \( Q \) be a finite semilattice, so that it is a lattice with meet

\[
q \land_{Q} q' = \bigvee_{Q} \{ r : r \leq_{Q} q \text{ and } r \leq_{Q} q' \}
\]
Then \( q \in Q \) is join-irreducible (resp. meet-irreducible) if (i) \( q \neq \bot_Q \) (resp. \( q \neq \top_Q \)) and (ii) whenever \( q = r \lor_Q r' \) (resp. \( q = r \land_Q r' \)) then \( q = r \) or \( q = r' \). Let \( J(Q) \), \( M(Q) \subseteq Q \) be the sets of join-irreducible and meet-irreducible elements of \( Q \).

**Lemma 2.15 (see [2])** The categories of finite semilattices and finite strict closure spaces are equivalent. Indeed, the following functor \( G : \text{JSL}_f \to \text{Cl}_f \) is an equivalence:

\[
GQ = (J(Q), \text{cl}_Q) \quad \text{where} \quad \text{cl}_Q(A) = \{ j \in J(Q) : j \leq_Q \bigvee_Q A \}
\]

\[
Gf = \{(j,j') \in J(Q) \times J(Q') : j' \leq_Q f(j)\} : GQ \to GQ'
\]

for any semilattice homomorphism \( f : Q \to Q' \).

**Remark 2.16** The associated equivalence \( C : \text{Cl}_f \to \text{JSL}_f \) maps a finite strict closure space \( Z \) to the semilattice \( CZ \) of all closed subsets of \( Z \) w.r.t. inclusion order. Its bottom is \( \emptyset \) and it has joins \( A \lor_{CZ} B = \text{cl}_Z(A \cup B) \) (the meet being intersection). A continuous relation \( B : Z_1 \to Z_2 \) is mapped to \( CB : CZ_1 \to CZ_2, \quad CB(A) = \text{cl}_Z(B[A]) \).

The equivalence \( G \) lifts to an equivalence of automata, assigning to each \( \text{JSL-dfa} \) a corresponding ‘nondeterministic closure automaton’ in \( \text{Cl}_f \).

**Definition 2.17** (a) A nondeterministic closure automaton (nca) is a triple \( N = (Z, R_a, F) \) where \( Z \) is a finite strict closure space (of states), the transition relations \( R_a \subseteq Z \times Z \) are continuous for \( a \in \Sigma \), and the final states \( F \subseteq Z \) form an open set. Given another nca \( N' = (Z', R'_a, F') \), a morphism \( B : N \to N' \) is a continuous relation \( B : Z \to Z' \) such that \( R'_a \circ B = B \circ R_a \) for each \( a \in \Sigma \), and \( z \in F \) iff \( B[z] \cap F' \neq \emptyset \). The category of ncas (and the above morphisms with continuous composition) is denoted \( \text{Nca} \).

(b) A pointed nca \( (N, I) \) is an nca \( N = (Z, R_a, F) \) equipped with a closed subset \( I \subseteq Z \) of initial states. Morphisms \( B \) between pointed ncas must additionally satisfy \( \text{cl}_{Z'}(B[I]) = I' \). The category of pointed ncas is denoted \( \text{Nca}_* \).

(c) The language accepted by a pointed nca \( (N, I) \) is the set \( L_{N}(I) \subseteq \Sigma^* \) of words \( w \) such that some state in \( I \) has some \( w \)-path to a final state.

**Remark 2.18** (1) Every nfa \( N = (Z, R_a, F) \) may be viewed as an nca where \( Z \) is discrete, i.e., it has the identity closure \( \text{cl}_Z = id_{Z} \). This nca is well-defined because (i) every relation between discrete closure spaces is continuous and (ii) every subset of a discrete closure space is both open and closed. Therefore we have full embeddings \( \text{Nfa} \hookrightarrow \text{Nca} \) and \( \text{Nfa}_* \hookrightarrow \text{Nca}_* \).

(2) Every (pointed) nca has an underlying (pointed) nfa where we forget the closure. In contrast to the previous statement, this does not define functors \( \text{Nca} \to \text{Nfa} \).
and $\text{Nca}_* \to \text{Nfa}_*$ because composition of $\text{Nca}$-morphisms is not the relational composition. Note that the language $L_N(I)$ accepted by a pointed nca $(N, I)$ is the language accepted by its underlying pointed nfa.

Lemma 2.19 (see [2]) The categories of (pointed) JSL-dfas and (pointed) ncas are equivalent. Indeed, the equivalence $G : \text{JSL}_f \to \text{Cl}_f$ described above lifts to equivalences:

$$G : \text{Jdfa} 	o \text{Nca} \quad \text{with} \quad G(Q, \delta_a, F) = (GQ, G\delta_a, F')$$

$$G_* : \text{Jdfa}_* \to \text{Nca}_* \quad \text{with} \quad G_*(Q, \delta_a, F, q_0) = (GQ, G\delta_a, F', I_{q_0})$$

where $F' = J(Q) \cap F$ and $I_{q_0} = \{ q \in J(Q) : q \leq Q q_0 \}$. On morphisms we have $Gf = Gf$ and $G_* f = G f$.

Proof. [Sketch] This follows from Lemma 2.15. Briefly, $G : \text{JSL}_f \to \text{Cl}_f$ defines an equivalence and one can apply it to the carrier $Q$ and each homomorphism $\delta_a$ of a JSL-dfa. Furthermore, the final states arise as a morphism $Q \to 2$ and initial states as a morphism $2 \to Q$, so one may again apply $G$. The resulting structure is the equivalent (pointed) nca. \[ \square \]

Hence JSL-dfas and ncas are essentially the same structures, although the latter have the significant advantage of having fewer states. For example, if a JSL-dfa has free carrier, then the corresponding nca is exponentially smaller. The languages accepted by JSL-dfas and ncas are by definition just the languages accepted by their underlying dfas and nfas, respectively, and are preserved by the equivalence:

Lemma 2.20 A pointed JSL-dfa accepts the same language as its equivalent pointed nca, i.e., $L_D(q) = L_N(I)$ where $(N, I) = G_*(D, q)$.

Proof. Let $D = (Q, \delta_a, F)$ and recall $I = J(Q) \cap \downarrow_Q q$, where $\downarrow_Q q = \{ q' \in Q : q' \leq q \}$. Then $w \in L_D(q)$ iff $\delta_w(q) \in F$ by definition. Equivalently $w \in L_D(q)$ iff $f \circ \delta_w \circ i = \text{id}_2$ where the JSL$_f$-morphism $i : 2 \to Q$ is defined $i(1) = q$ (and, necessarily, $i(0) = \bot_Q$) and the JSL$_f$-morphism $f : Q \to 2$ is defined $f(q) = 1$ iff $q \in F$, recalling that $F$ is a prime filter.

Now $f \circ \delta_w \circ i = \text{id}_2$ iff $Gf \bullet G\delta_w \bullet Gi = G\text{id}_2 = \text{id}_{\{1\}}$ (where $\{1\}$ has identity closure) because $G$ is faithful, being an equivalence. Observe that $Gi[1] = J(Q) \cap \downarrow_Q q = I$ and also $Gf \subseteq J(Q) \times \{1\}$ is such that $Gf[j] = \{1\}$ iff $j \in F$. We now show

$$Gf \bullet G\delta_w \bullet Gi = \text{id}_{\{1\}} \quad (1)$$

iff there exists $j \in I$ such that $G\delta_w[j] \cap F \neq \emptyset$. The latter is equivalent to saying that $GD$ accepts $w$ via the initial states $I$.
Assuming acceptance implies (1) because the relation $G\delta_w \cdot G_i$ contains $G\delta_w \circ G_i$. Conversely, assuming (1), first observe that:

$$G\delta_w \cdot G_i[1] = \text{cl}_Q(G\delta_w \circ G_i[1]) = \text{cl}_Q(\bigcup_{j \in I} \{j_0 \in J(Q) : j_0 \leq_Q \delta_w(j)\}) = \{j' \in J(Q) : j' \leq \bigvee_Q \{\delta_w(j) : j \in I\}\}.$$

Then (1) implies that $G\delta_w \cdot G_i[1]$ has non-empty intersection with $F$, i.e., there exists $j' \leq_Q \bigvee_Q \{\delta_w(j) : j \in I\}$ such that $j' \in F \cap J(Q)$, so in particular $j' \neq \bot_Q$. Since $F$ is upwards closed, there is some non-zero $\delta_w(j) \in F$ (where $j \in I$) and hence also some non-zero join-irreducible beneath $\delta_w(j)$ lies in $F$. This implies $G\delta_w[j] \cap F$ is non-empty. \qed

## 3 Reversal, Reachability and Minimality

The purpose of the present section is to prove that every regular language $L$ has an associated minimal pointed nca $N(L)$ accepting $L$, which is unique up to isomorphism. We also present a construction of this minimal pointed nca, which is analogous to Brzozowski’s classical construction of the minimal pointed dfa [6] (see also [4] for a (co-)algebraic view). Recall that the latter takes any pointed nfa $(N,I)$ accepting $L$ and constructs $L$’s minimal dfa as follows:

$$\text{reach} \circ \text{rev} \circ \text{reach} \circ \text{rev}(N,I)$$

(2)

Here $\text{rev}$ reverses transitions and also swaps the final and initial states,

$$(N,I) = (Z,R_a,F,I) \implies \text{rev}(N,I) = (Z,\tilde{R}_a,I,F),$$

where $\tilde{R}_a$ denotes the converse of the relation $R_a$. Furthermore, $\text{reach}$ performs the reachable subset construction, i.e., it forms the subset dfa and takes its reachable part. In this section we introduce these two operations for pointed ncas. We then prove that the minimal pointed nca $N(L)$ arises in exactly the same way as (2), only now taking any pointed nca accepting $L$ as input. In particular, any pointed nfa will do.

The above nfa operation $\text{rev}$ extends to a self-duality of the category $\text{Nfa}_*$ of pointed nfas, defined on objects as above and on morphisms by $B \mapsto \tilde{B}$. To see that it works on the final/initial states, let $\mathcal{F} = \{(z,+) : z \in F\} \subseteq Z \times \mathbf{1}$ and $\mathcal{I} = \{(+,z) : z \in I\} \subseteq \mathbf{1} \times Z$ where $\mathbf{1} = \{+\}$. Then we can rewrite our conditions on $\text{Nfa}_*$-morphisms (see Definition 2.2) as $\mathcal{F}' \circ B = \mathcal{F}$ and $B \circ \mathcal{I} = \mathcal{F}'$, which clearly dualize under converse. Therefore in order to generalize $\text{rev}$ to pointed ncas, we describe a suitable self-duality of $\text{Nca}_*$. It is based on the well-known self-duality of $\text{JSl}_f$:

**Lemma 3.1** The following functor $D : \text{JSl}_f^{op} \to \text{JSl}_f$ is an equivalence: on objects let $DQ = Q^{op}$ (which has carrier $Q$, bottom $\top_Q$ and join $\wedge_Q$) and on morphisms
$f : Q_1 \to Q_2$ define $Df^\text{op} : Q_2^\text{op} \to Q_1^\text{op}$ by

$$Df^\text{op}(q_2) = \bigvee_{q_1 \in Q_1} \{q_1 \in Q_1 : f(q_1) \leq Q_2 q_2\}.$$  

**Proof.** [Sketch] The self-duality of $\text{JSL}_f$ follows from the adjoint functor theorem for posets. Finite join-semilattices are finite posets with all joins (= colimits) and join-semilattice morphisms are monotone maps that preserve all joins. Consequently, each $f : Q_1 \to Q_2$ has a right adjoint $g : Q_2^\text{op} \to Q_1^\text{op}$ where the order is reversed because right adjoints preserve all meets. The uniqueness of adjoints implies that this is an equivalence. Its explication yields the above action on the morphisms. \qed

Since $\text{JSL}_f$ is equivalent to $\text{Cl}_f$ (see Lemma 2.15 and Remark 2.16), it follows that $\text{Cl}_f$ is also self-dual, with dual equivalence

$$D = (\text{Cl}_f^\text{op} \xrightarrow{\text{op}} \text{JSL}_f^\text{op} \xrightarrow{D} \text{JSL}_f \xrightarrow{G} \text{Cl}_f).$$

We now describe this self-duality explicitly. Recall that $CZ$ denotes the semilattice of closed subsets of a finite strict closure space $Z$, and that $M(CZ)$ is the set of meet-irreducibles of $CZ$.

**Proposition 3.2** For any finite strict closure space $Z$ we have

$$\text{DZ} = M(CZ) \quad \text{cl}_{\text{DZ}}(X) = \{A \in M(CZ) : \bigcap X \subseteq A\},$$

and for any $B : Z_1 \to Z_2$, the continuous relation $\text{DB}^\text{op} : M(CZ_2) \to M(CZ_1)$ consists of all those $(A_2, A_1) \in M(CZ_2) \times M(CZ_1)$ such that:

$$B[A] \subseteq A_2 \quad \implies \quad A \subseteq A_1 \quad \text{for every } A \in J(CZ_1).$$

**Proof.** We have $\text{DZ} = G((CZ)^\text{op}) = (M(CZ), \text{cl})$ where $M(CZ) = J((CZ)^\text{op}) \subseteq Q$ are the meet-irreducibles and:

$$\text{cl}(S) = \{j \in J((CZ)^\text{op}) : j \leq_{(CZ)^\text{op}} \bigvee_{(CZ)^\text{op}} S\}$$

$$= \{m \in M(CZ) : \bigwedge_{CZ} S \leq_{CZ} m\}$$

$$= \{m \in M(CZ) : \bigcap S \subseteq m\}$$

using the fact that the meet in $CZ$ is intersection. Likewise every $\text{JSL}_f$-morphism $f : Q \to Q'$ has a dual morphism $Df^\text{op} : Q'^\text{op} \to Q^\text{op}$ defined $Df^\text{op}(q') = \bigvee_{Q} f^{-1}(\downarrow_{Q} q')$. Therefore every continuous relation $B : Z \to Z'$ has a dual continuous relation $\text{DB}^\text{op} : \text{DZ}' \to \text{DZ}$ defined as follows: let $f = CB$ be the $\text{JSL}_f$-morphism corresponding to $B$ and then apply the equivalence $G : \text{JSL}_f \to \text{Cl}_f$ to the homomorphism
\(D \langle f \rangle\). Then

\[
\text{D}B = \{(j', j) \in J(Q) \times J(Q) : j \leq Q M \langle j' \rangle\}
\]

\[
= \{(m', m) \in M(Q) \times M(Q) : \forall j \in J(Q). (f(j) \leq Q M' \Rightarrow j \leq Q M)\}
\]

\[
= \{(m', m) : \forall j \in J(Q). (\text{cl}_{Q}(B[j]) \leq Q M' \Rightarrow j \leq Q M)\}
\]

In the last step we use that \(B[j]\) is closed since \(B\) is continuous.

Given any pointed nca \((Z, R_a, F, I)\), the previous duality naturally leads to a pointed nca with states \(M(CZ)\) by applying \(D\) to (a) \(R_a : Z \rightarrow Z\), (b) \(F\) considered as a continuous relation \(F : Z \rightarrow \{1\}\) and (c) \(I\) considered as a continuous relation \(I : \{1\} \rightarrow Z\). This is the reversal of pointed ncas: by applying \(D\) to \(F\) we get the subset \(I^d \subseteq M(CZ)\) of all \(A \in M(CZ)\) containing \(Z \setminus F\). By applying \(D\) to \(I\) we get the subset \(F^d \subseteq M(CZ)\) of all \(A \in M(CZ)\) with \(I \not\subseteq A\).

**Definition 3.3** The reversal of a pointed nca is defined by

\[
\text{rev}(Z, R_a, F, I) = (M(CZ), D \langle R_a \rangle, F^d, I^d).
\]

**Example 3.4** Take any pointed nfa and view it as a pointed nca with identity closure. Then \(CZ = \mathcal{P}Z\) has meet-irreducibles \(Z \setminus \{z\}\) for \(z \in Z\) and the reversal is the classical nfa reversal, modulo the bijection \(z \mapsto Z \setminus \{z\}\).

**Theorem 3.5** The category \(\text{Nca}\) is self-dual: the object map \(\text{rev}\) extends to an equivalence \(\text{rev} : \text{Nca} \rightarrow \text{Nca}\). It assigns to every morphism \(B : (Z, R_a, I, F) \rightarrow (Z', R'_a, I', F')\) the morphism \(\text{rev}(B) \subseteq M(CZ) \times M(CZ)\) of all pairs \((A', A)\) such that

\[
B[X] \subseteq A' \implies X \subseteq A \quad \text{for every } X \in J(CZ).
\]

**Proof.** In Definition 3.3 we defined the object part of the dual equivalence \(\text{rev} : \text{Nca} \rightarrow \text{Nca}\). We now prove that it actually defines a functor.

(i) The action on continuous relations \(R_a\) and \(\text{Nca}\)-morphisms \(B\) is the action of \(D\), see Proposition 3.2.

(ii) The initial states form a closed set \(I \subseteq CZ\), or equivalently a join-semilattice morphism \(i : 2 \rightarrow CZ\) such that \(i(1) = I\). Dually we have the join-semilattice morphism \(i' = Di : (CZ)^\ast \rightarrow 2^\ast\) defined by

\[
i'(x) = \bigvee_2 i^{-1}(\downarrow_{CZ} x) = \begin{cases} \top_2 = 0 & \text{if } i(1) \not\subseteq CZ x \\ 1 & \text{otherwise} \end{cases}
\]
Therefore the new final states are:

\[
F^d = J((CZ)^{op}) \cap i'^{-1}(\top_{2^{op}}) \\
= \{ m \in M(CZ) : I \not\subseteq_{CZ} m \} \\
= \{ m \in M(CZ) : I \not\subseteq m \}.
\]

(iii) The final states \( F \) form an open set in \((Z, \text{cl}_Z)\), or equivalently a join-semilattice morphism \( f : CZ \to 2 \) such that \( Z \setminus F = \bigvee_{CZ} f^{-1}(\{0\}) \). The dual join-semilattice morphism \( f' = Df^{op} : 2^{op} \to (CZ)^{op} \) is defined by \( f'(b) = \bigvee_{CZ} f^{-1}(\{0\}) \). Consequently:

\[
I^d = \{ j \in J((CZ)^{op}) : j \leq_{(CZ)^{op}} f'(\top_{2^{op}}) \} \\
= \{ m \in M(CZ) : \bigvee_{CZ} f^{-1}(\{0\}) \leq_{CZ} m \} \\
= \{ m \in M(CZ) : Z \setminus F \subseteq m \}.
\]

\[\Box\]

**Proposition 3.6** If a pointed nca accepts \( L \) then its reverse pointed nca accepts the reversed language \( \text{rev}(L) \).

**Proof.** A pointed JSL-dfa \((D, q) = (Q, \delta_a, F, q)\) accepts a word \( w \) iff \( f \circ \delta_w \circ i = \text{id}_2 \), where \( f : Q \to 2 \) represents the final states, \( \delta_w \) is the inductive extension of the \( \delta_a \)'s and \( i : 2 \to Q \) represents \( q \in Q \). Now \( J\text{dfa} \) is self-dual (because \( N\text{ca} \) is) with dual pointed JSL-dfa \((Q^{op}, D\delta_a^{op}, \uparrow Q^{op} q, \bigvee Q(Z \setminus F))\). The dual of the equality above is \( Df^{op} \circ D\delta_w^{op} \circ f' = \text{id}_{2^{op}} \) and furthermore \( Di \) corresponds to the final states in the dual machine, just as \( Df \) corresponds to the initial states. Consequently \((D, q)\) accepts \( w \) iff its dual machine accepts the reversed word \( w^r \). This property is inherited by \( N\text{ca} \) because the equivalence \( G_* : J\text{dfa} \to N\text{ca} \) preserves languages, see Lemma 2.20. \[\Box\]

Next we extend the operation \text{reach} from nfas to pointed ncas. A pointed nfa is reachable if each state is reached from some initial state by transitions. Equivalently, the pointed nfa has no proper sub nfas. Here ‘sub nfa’ refers to the category \( N\text{fa} \), i.e. \( N' \) is a sub nfa of \( N \) if there is a morphism \( B : N' \to N \) where \( B \) is an injective function. Implicitly one uses the (onto relation, injective function) factorization system in \( \text{Rel}_f \) and lifts it to \( N\text{fa} \). Similar remarks apply to sub dfas: (i) they arise as injective dfa morphisms via the (surjection, injection) factorization system of \( \text{Set}_f \), (ii) a pointed dfa is reachable iff it has no proper sub dfas.

In order to define \text{reachable pointed ncas}, we first need the appropriate concept of \text{sub nca}. To this end, we take the (epi, mono) = (surjection, injection) factorization system of \( J\text{SL}_f \) and apply the equivalence of \( J\text{SL}_f \) and \( \text{Cl}_f \) to obtain a corresponding factorization system in \( \text{Cl}_f \).

**Lemma 3.7** Every continuous relation \( B : Z_1 \to Z_2 \) has an essentially unique (epi,
monomorphism in $\text{Cl}_f$. Moreover, $B$ is monic (resp. epic) iff the function

$$CB : CZ_1 \to CZ_2, \quad CB(S) = \text{cl}_{Z_2}(B[S]),$$

is injective (resp. surjective).

**Proof.** The functor $C : \text{Cl}_f \to \text{JSL}_f$ preserves and reflects monos and epis, being an equivalence. \hfill \Box

**Definition 3.8** (a) A pointed nca $(N', I')$ is a sub nca of $(N, I)$ if there exists an $\text{Nca}_\ast$-monomorphism $m : (N', I') \to (N, I)$.

(b) A pointed nca $(N', I')$ is a quotient nca of $(N, I)$ if there exists an $\text{Nca}_\ast$-epimorphism $e : (N, I) \to (N', I')$.

(c) A pointed nca is called reachable if it has no proper sub ncas, simple if it has no proper quotient ncas, and minimal if it is both reachable and simple.

**Proposition 3.9** Any sub or quotient nca of $(N, I)$ accepts the same language $L_N(I)$.

**Proof.** Viewed as their equivalent pointed JSL-dfa, we have injective or surjective deterministic automata morphisms which preserve the initial state. These certainly preserve the language and by Lemma 2.20 the respective pointed ncas accept the same languages. \hfill \Box

To obtain a more concrete characterization of reachability and simplicity, we shall restrict to ncas whose closure is normalized in the following sense:

**Lemma 3.10** Every finite strict closure space $Y$ is isomorphic to another finite strict closure space $Z$ such that:

(i) $Z$ is separable, that is, $z \neq z' \in Z$ implies $\text{cl}_Z(z) \neq \text{cl}_Z(z')$.

(ii) $S \in CZ$ is join-irreducible in $CZ$ iff $S = \text{cl}_Z\{z\}$ for some $z \in Z$.

**Proof.**

(i) Recall from Lemma 2.15 and Remark 2.16 the equivalence $G : \text{JSL}_f \to \text{Cl}_f$ and its associated equivalence $C : \text{Cl}_f \to \text{JSL}_f$. Then $Y$ is isomorphic to the closure space $Z = GCY$ whose carrier $J(CY)$ is the set of join-irreducibles in $CY$ and whose closure is defined by $\text{cl}_{GY}(S) = \{j \in J(CY) : j \subseteq \text{cl}_Y(\bigcup S)\}$ for any $S \subseteq J(CY)$. This closure space $GY$ is separable: given distinct join-irreducibles $j \neq j' \in J(CY)$ then wlog $j \nsubseteq CY j'$, hence $j \nsubseteq j' = \text{cl}_Y(j')$ and $j \notin \text{cl}_{GY}\{j\}$. But clearly $j' \in \text{cl}_{GY}\{j\}$. But clearly $j' \in \text{cl}_{GY}\{j\}$.

(ii) In any closure space $Z'$, every join-irreducible in $CZ'$ is the closure of some singleton set. For if $S = \text{cl}_{Z'}(S)$ is join-irreducible and $S = S_1 \cup S_2$ then $S = \text{cl}_{Z'}(S_1) \cup_{CZ'} \text{cl}_{Z'}(S_2)$ and hence wlog $S = \text{cl}_{Z'}(S_1)$. Continuing we get either $S = \text{cl}_{Z'}(\emptyset) = \emptyset$ (a contradiction) or $S$ is the closure of a singleton set. For the particular closure space $GY = (J(CY), \text{cl}_{CY})$ we also have the converse, i.e., the closure of a singleton subset of $J(CY)$ is join-irreducible.
in \(C(GCY)\). This follows because the closed sets of \(GCY\) take the form \(J(CY) \cap ^C_{CY} S\) for some \(S \in CY\) and in particular the closure of a singleton \(\{j\}, j \in J(CY)\), consists of all join-irreducibles smaller than or equal to \(j\). It follows that if \(cl_{GY}(\{j\}) = K_1 \lor_{C(GCY)} K_2\) then some \(K_i\) contains \(j\) by join-irreducibility of \(\{j\}\), wlog \(j \in K_1\). Then \(cl_{GY}(\{j\}) \subseteq cl_{GY}(K_1) = K_1\), and the converse is clear.

\[\square\]

**Definition 3.11** A (pointed) nca is **normalized** if its closure satisfies the conditions of Lemma 3.10.

**Corollary 3.12** Every nca is isomorphic to a normalized nca.

**Proposition 3.13** A normalized pointed nca \((Z, R_a, F, I)\) is reachable iff for every \(z \in Z\) there exists a word \(w_z \in \Sigma^*\) such that:

(i) There is a \(w_z\)-path from some initial state to \(z\) in the underlying nfa.

(ii) Every \(w_z\)-path from every initial state terminates at an element of \(cl_Z(z)\).

**Proof.** Suppose \((N, I)\) is a reachable pointed nca. Then by the equivalence of \(G_s : Jdfa_s \to Nca_s\), we have a corresponding pointed JSL-dfa \((D, I) = (CZ, CR_a, F', I)\). Its final states \(F' \subseteq CZ\) are defined \(F' = \{A \in CZ : A \cap F \neq \emptyset\}\). Note \(I \in CZ\) is now a single state. By the equivalence of \(Nca_s\) and \(Jdfa_s\), we know that \((D, I)\) has no proper subobjects i.e. every injective \(Jdfa_s\)-morphism into \((D, I)\) is bijective. Viewing \(D\) as its underlying dfa, one can list those states reachable from the state \(I \in CZ\) via transitions and then construct the join-subsemilattice of \(CZ\) generated by this set. This defines a pointed sub JSL-dfa, using the fact that the transition functions \(CR_a : CZ \to CZ\) are JSL-\(f\)-morphisms, hence an injective \(Jdfa_s\)-morphism which is necessarily bijective. It follows that every \(A \in CZ\) arises as a join of elements reachable from the single state \(I\) via transitions. In particular the join-irreducible elements must be reachable from \(I\) via transitions, since they form the minimal generating set. So take any element \(z \in Z\) and consider its closure \(A = cl_Z(z)\), this being an element of \(D\)’s carrier. By our assumption that the closure has been normalized, \(A\) is join-irreducible in \(CZ\). Therefore there exists some \(w_z \in \Sigma^*\) such that \(CR_{w_z}(I) = A\) i.e. we have a deterministic \(w_z\)-path in \(D\) from the initial state \(I\) to the state \(A\). Then \(A = cl_Z(R_{w_z}[I])\) and since \(A\) is join-irreducible and \(cl_Z\) is separable we deduce \(z \in R_{w_z}[I]\), i.e., the first condition holds. The second condition follows because \(z' \in R_{w_z}[z_0] \) implies \(z' \in CR_{w_z}(I) = A = cl_Z(z)\).

Conversely, suppose the two conditions hold for some pointed nca \((Z, R_a, F, I)\). Consider its equivalent pointed JSL-dfa with carrier \(CZ\). The conditions imply that every join-irreducible in \(CZ\) is reachable from the single state \(I\). We can form a sub JSL-dfa by closing under the transitions and then forming the generated subalgebra. Since every join-irreducible is reachable, this gives the original JSL-dfa. Furthermore this is the smallest sub JSL-dfa, which implies the respective pointed nca is reachable.

\[\square\]

**Remark 3.14** The above condition (ii) may be felt surprising. However, recall...
that reachability for pointed ncas was defined by complete analogy with nfas: no proper sub nca exists. For pointed dfas (viewed as ncas with identity closure) this is the usual notion of reachability, since there is exactly one \( w_z \)-path from the unique initial state. However the same cannot be said for pointed nfas.

**Proposition 3.15** Viewed as a pointed nca with identity closure, a pointed nfa is reachable iff its reachable subset construction (that is, the reachable part of the subset dfa) contains all singleton sets.

**Proof.** Let \((Z, R, F, I)\) be a pointed nfa. If every singleton subset lies in its reachable subset construction, then every \( z \in Z \) has some \( w_z \in \Sigma^* \) such that the unique path from the single state \( I \) terminates at \( z \). Since the path is unique and \( z \in \text{cl}_Z(z) = \{z\} \), it follows that we have a reachable pointed nca by Proposition 3.13. Conversely, suppose this pointed nfa defines a reachable pointed nca with identity closure. Then by Proposition 3.13 each \( z \in Z \) has some \( w_z \) such that \( I \xrightarrow{w_z} \{z\} \) because no other state lies in the closure of \( \{z\} \).

We now provide further properties of reachable pointed ncas.

**Proposition 3.16** Suppose one has a normalized reachable pointed nca accepting \( L \), then:

(i) Its underlying pointed nfa is reachable.

(ii) Its individual states accept derivatives of \( L \).

(iii) Varying the (closed) set of initial states, the languages it accepts are precisely the unions of \( L \)'s derivatives.

**Proof.** The first statement follows immediately from Proposition 3.13 via the first condition (i). The second statement follows because for each \( z \in Z \), its closure is reachable from \( I \) in the equivalent pointed JSL-dfa. Thus this closed set accepts a derivative of \( L \) and this language is preserved by the equivalence (see Lemma 2.20). Finally, for (iii) observe that the JSL-dfa equivalent to \( N \) accepts precisely the unions of derivatives of \( L \) when varying the initial state: indeed, the states reachable via transitions accept precisely the derivatives of \( L \), and all other states arise as a join of such states. Since languages are preserved by the equivalence, (iii) follows.

Every pointed nfa has a smallest sub nfa, which is necessarily reachable. That is, one simply discards all those states not reachable from the initial states by transitions. From the categorical standpoint this means that the intersection of all pointed sub nfas exists. Similarly, for any pointed nca, the intersection of all pointed sub ncas exists i.e. we can always construct a unique reachable sub nca.

**Definition 3.17** The reachable part \( \text{reach}(N, I) \) of a pointed nca \((N, I)\) is the unique reachable pointed sub nca, i.e. the intersection of all pointed sub ncas.

**Notation 3.18** Given any nca with carrier \( Z \), any word \( w \in \Sigma^* \) and any subset \( I \subseteq Z \), we write \( z \xrightarrow{w} z' \) whenever there is a \( w \)-path from \( z \) to \( z' \) in the underlying nfa. Then \( w \cdot I \in CZ \) denotes the closure of \( \{z' \in Z : \exists z \in I. z \xrightarrow{w} z'\} \).
Proposition 3.19 \( \text{reach}(Z, R_a, F, I) \) is isomorphic to \( (Z', R'_a, F', I') \) where:

(i) \( Z' \subseteq CZ \) is the set of \( w \cdot I \)'s not arising as joins of other \( v \cdot I \)'s in \( CZ \);
(ii) \( R'_a = \{(u \cdot I, v \cdot I) \in Z' \times Z' : v \cdot I \subseteq u a \cdot I \} \) for each \( a \in \Sigma \);
(iii) \( F' = \{A \in Z' : F \cap A \neq \emptyset \} \);
(iv) \( I' = \{A \in Z' : A \subseteq I \} \).

Proof. [Sketch] This follows by considering equivalent pointed JSL-dfa i.e. we close under the deterministic transitions from \( I \), form the generated subalgebra, and then convert this JSL-dfa back into a pointed nca.

Remark 3.20 Applying this to an nfa (i.e. a pointed nca with identity closure), one finds that \( \text{reach}(N, I) \) is never larger than the reachable subset construction \( \{w \cdot I : w \in \Sigma^*\} \).

Next we characterize simple pointed ncas. Recall that a dfa is simple iff distinct states accept distinct languages. Analogously:

Proposition 3.21 A pointed nca \( (N, I) \) with carrier \( Z \) is simple iff distinct closed subsets accept distinct languages i.e. if \( A \neq B \in CZ \) then \( L_N(A) \neq L_N(B) \).

Proof. Let \( D \) be the JSL-dfa equivalent to \( N \). It is simple since \( N \) is, so the unique map into the final JSL-da \( D_{\text{Reg}^*(\Sigma)} \) (see Example 2.7) is injective. This means that distinct states of \( D \) accept distinct languages, hence the statement follows from Lemma 2.20.

By Theorem 3.5 we know \( \text{Nca}_* \) is self-dual. Moreover reachable and simple are dual concepts, see Definition 3.8. Then if \( (N, I) \) is any pointed nca accepting \( L \), it follows that

\[ \text{sim}(N, I) \overset{\text{def}}{=} \text{rev} \circ \text{reach} \circ \text{rev}(N, I) \]

is a simple pointed nca accepting \( L \) i.e. the simplification of \( (N, I) \). Categorically, it is the cointersection of all quotient ncas.

Next consider \( \text{reach} \circ \text{sim}(N, I) \) which is certainly reachable. Importantly it is also simple, using Proposition 3.21 and the fact \( \text{reach}(N, I) \) is a sub nca of \( (N, I) \). Then by definition it is a minimal pointed nca accepting \( L \). In fact:

Proposition 3.22 For every regular languages \( L \), there is up to isomorphism only one minimal pointed nca accepting \( L \).

Proof. This follows from a more general result in [1, Lemma 3.22] since the minimal nca is an instance of a well-pointed coalgebra. Let us briefly sketch the argument. Suppose one has two minimal pointed ncas accepting \( L \). Equivalently one has two minimal pointed finite JSL-dfas accepting \( L \), where minimal now means reachable and simple in Jdfa_*.. Each one has a unique Jda-morphism to the deterministic JSL-automaton \( D_{\text{Reg}^*(\Sigma)} \) of regular languages (see Example 2.7(c)), assigning to each state the language it accepts. These morphisms factorize into a surjective morphism followed by an inclusion i.e. their image defines a sub dfa of \( D_{\text{Reg}^*(\Sigma)} \). Since they are both simple, they are each isomorphic to their respective image. Since they are
both reachable, by Proposition 3.16 the carrier of this image is precisely the set of unions of derivatives of \( L \). Therefore they are isomorphic to each other.

**Notation 3.23** Let \( Q_L \) denote the semilattice of all unions of derivatives of \( L \).

**Proposition 3.24** The following pointed nca \( \mathcal{N}(L) \) for is minimal nca for \( L \):

(i) States \( Z_L = J(Q_L) \) i.e. the non-empty derivatives of \( L \) that are not unions of others derivatives, endowed with the closure:

\[
\text{cl}_L(A) = \{ K \in Z_L : K \subseteq \bigcup A \},
\]

(ii) transitions \( R_a = \{ (K, K') \in Z_L \times Z_L : K' \subseteq a^{-1} K \} \) for each \( a \in \Sigma \),

(iii) as final states those \( K \in Z_L \) containing \( \varepsilon \),

(iv) as initial states those \( K \in Z_L \) which are subsets of \( L \).

**Proof.** It is easy to see that the minimal pointed JSL-dfa accepting \( L \) is the finite subautomaton of \( D_{\text{Reg}}(\Sigma) \) (see Example 2.7(c)) generated by \( L \). Hence it has states \( Q_L \), transitions \( K \rightarrow a^{-1} K \) \( (K \in Q_L) \), initial state \( L \), and the final states are precisely those languages in \( Q_L \) containing \( \varepsilon \). Now apply the equivalence of Jdfa and Nca.

Then the main result in this section follows:

**Theorem 3.25** For any pointed nca \( (\mathcal{N}, I) \) accepting \( L \), the pointed nca:

\[
\text{reach} \circ \text{rev} \circ \text{reach} \circ \text{rev}(\mathcal{N}, I)
\]

is a minimal pointed nca accepting \( L \), and is hence isomorphic to \( \mathcal{N}(L) \).

**Proof.** By Proposition 3.6, the dual of a pointed nca accepts the reversed language. Since \( \text{reach} \) preserves the accepted language, the above pointed nca accepts \( L \). So by Proposition 3.22 it suffices to show it is a minimal pointed nca. It is clearly reachable. Moreover, \( \text{rev} \circ \text{reach} \circ \text{rev}(\mathcal{N}, I) \) is the dual of a reachable pointed nca and hence is simple. Finally, \( \text{reach} \) preserves simplicity (as previously explained), so we are done.

Finally, since Nca is self-dual and reachability and simplicity are dual:

**Proposition 3.26** For each regular language \( L \), the reverse of the minimal pointed nca \( \mathcal{N}(L) \) is isomorphic to the minimal pointed nca for \( \text{rev}(L) \), shortly,

\[
\text{rev}(\mathcal{N}(L)) \cong \mathcal{N}(\text{rev}(L)).
\]

**Proof.** By Proposition 3.6 we know the dual of \( \mathcal{N}(L) \) accepts \( \text{rev}(L) \). Furthermore minimality is a self-dual property because ‘no proper subobjects’ and ‘no proper quotients’ dualize. Hence, the dual of \( \mathcal{N}(L) \) is minimal and is isomorphic to \( \mathcal{N}(\text{rev}(L)) \) by Proposition 3.22.
4 State-minimal Nondeterministic Automata

Each regular language \( L \) is accepted by the underlying nfa of its minimal pointed nca \( \mathcal{N}(L) \). Although this nfa is never larger than the minimal dfa, it generally need not be a state-minimal nondeterministic acceptor [7]. In this section we present a natural sufficient condition for state minimality. Recall that a strict closure \( \text{cl}_L \) is topological if it is induced by a topology on \( Z \), i.e., it satisfies the equation

\[
\text{cl}_L(A \cup B) = \text{cl}_L(A) \cup \text{cl}_L(B) \quad \text{for all} \; A, B \subseteq Z.
\]

Definition 4.1 An nca is topological if its closure is topological.

Lemma 4.2 \( \mathcal{N}(L) \) is topological iff the lattice \( Q_L \cong CZ_L \) is distributive.

Proof. If \( \text{cl}_L \) is topological then \( CZ_L \subseteq \mathcal{P}Z_L \) is closed under union and intersection, hence it is a distributive lattice. Conversely suppose that \( Q_L \cong CZ_L \) is distributive, and let \( A = \{K_1, \ldots, K_m\} \) and \( B = \{L_1, \ldots, L_n\} \) be closed subsets of \( Z_L \). We need to show that \( A \cup B \) is closed, so suppose that \( K \in \text{cl}_L(A \cup B) \), that is, \( K \subseteq K_1 \cup \cdots \cup K_m \cup L_1 \cup \cdots \cup L_n \) by the definition of \( \text{cl}_L \). Note that for any join-irreducible element \( x \) of a finite distributive lattice, \( x \leq y \lor z \) implies \( x \leq y \) or \( x \leq z \) (because \( x \leq y \lor z \) implies \( x = x \land (y \lor z) = (x \land y) \lor (x \land z) \), hence \( x = x \land y \) or \( x = x \land z \) since \( x \) is join-irreducible). We conclude that \( K \subseteq K_1 \cup \cdots \cup K_m \cup L_1 \cup \cdots \cup L_n \), so \( K \in \text{cl}_L(A) = A \) or \( K \in \text{cl}_L(B) = B \). Thus \( \text{cl}_L(A \cup B) \subseteq A \cup B \). \( \square \)

Example 4.3 (1) If \( \mathcal{N}(L) \) is an nfa (i.e., has identity closure) then it is topological. For example, this is the case whenever \( d_L = 2^{n_L} \), where \( d_L \) (resp. \( n_L \)) is the number of states of a state-minimal dfa (resp. nfa) accepting \( L \), see [2].

(2) If \( \mathcal{N}(L) \) is topological then so is \( \mathcal{N}(\text{rev}(L)) \). Indeed, recall that they are duals. Then since \( \mathcal{N}(L) \) is topological iff the closed sets \( CZ_L \) form a distributive lattice, its order-dual is also distributive and is isomorphic to \( CZ_{\text{rev}(L)} \).

(3) If \( \mathcal{N}(L) \) is topological and \( f : \Delta^* \to \Sigma^* \) is a surjective monoid morphism then \( \mathcal{N}(f^{-1}(L)) \) is also topological. Here one uses the fact that \( f^{-1} : \mathcal{P}\Sigma^* \to \mathcal{P}\Delta^* \) is an injective boolean morphism, providing an isomorphism between the semilattices \( Q_L \) and \( Q_{f^{-1}(L)} \).

(4) If \( L \) is intersection-closed (that is, each binary intersection of derivatives of \( L \) arises as a union of derivatives of \( L \)), then \( \mathcal{N}(L) \) is topological by Lemma 4.2. Examples of intersection-closed languages include:

(a) the languages \( \Sigma^* \) and \( \{w\} \) for \( w \in \Sigma^* \),
(b) the tail languages \((a + b)^*b(a + b)^{n-1} \) \((n \geq 1)\),
(c) linear codes, i.e., linear subspaces of the vector space \( \mathbb{Z}_2^n \) (viewed as languages over the binary alphabet);
(d) the languages \( L_f = \{w \in 2^n : f(w) = 1\} \subseteq 2^* \) where \( f : 2^n \to 2 \) is either the parity function, the majority function or any \( \mathbb{R} \)-weighted threshold function, i.e.,
\[
f(b_1, \ldots, b_n) = 1 \quad \text{iff} \quad \sum k_ib_i \geq t
\]
for some real-valued constants \( k_1, \ldots, k_n \) and \( t \).
(5) The language $L = a(a + b) + b(b + c)$ is not intersection-closed, yet $\mathcal{N}(L)$ is topological. Indeed, the derivatives of $L$ are $\emptyset$, $L$, $a + b$, $b + c$ and $\varepsilon$, and $(a + b) \cap (b + c) = b$ is not a union of derivatives. However, the lattice $Q_L$ is isomorphic to the lattice of all subsets of a two-element set, and hence distributive.

In [2] it was proved that $\mathcal{N}(L)$ is state-minimal provided that $L$ is intersection-closed. The following theorem is a generalization of this result, as witnessed by Example 4.3(4), (5) above.

**Theorem 4.4** Let $L$ be a regular language. If the minimal pointed nca $\mathcal{N}(L)$ is topological then its underlying nfa is state-minimal.

**Proof.** (1) By Lemma 2.19, the categories of pointed JSL-dfas and pointed non-deterministic closure automata are equivalent. In fact, $\mathcal{G}_*$ has the associated equivalence $\mathbb{C}_{\ast} : \text{Nca}_{\ast} \rightarrow \text{Jdfa}_{\ast}$ defined by

$$\mathbb{C}_{\ast}(Z, R, F, I) = (CZ, CR, F', I) \quad \text{and} \quad \mathbb{C}_{\ast}B = CB,$$

where $F' = \{ A \in CZ : A \cap F \neq \emptyset \}$. Here we use the equivalence $C : \text{Cl}_f \rightarrow \text{JSL}_f$, see Remark 2.16. Observe that $I \in CZ$ is now a single state in a JSL-dfa.

(2) Given any pointed nfa $(N, I) = (Z, R, F, I)$ accepting $L$. Viewing $(N, I)$ as a pointed nca $(\mathcal{N}, I)$ with identity closure and applying the above equivalence yields the JSL-dfa $(\mathcal{D}, I)$ where $\mathcal{D} = (\mathcal{P}Z, CR, F')$ is the subset construction. Let $Q = \{ L_D(q) : q \in \mathcal{P}Z \}$ be the set of languages accepted by the individual states in $\mathcal{D}$. Since these are precisely those languages accepted by the nfa $N$ (varying the set of initial states), we deduce that $Q$ is closed under finite unions and derivatives, so $Q$ is a semilattice of regular languages under $\emptyset$ and $\cup$. It forms the carrier of a JSL-dfa $\mathcal{D}' = (Q, \delta_a, \hat{F})$ with transitions $K \xrightarrow{a} a^{-1}K$ and final states $\hat{F} := \{ K \in Q : \varepsilon \in K \}$. Furthermore, $\mathcal{L}_D$ defines a surjective Jdfa*-morphism

$$\mathcal{L}_D : (\mathcal{D}, I) \rightarrow (\mathcal{D}', L),$$

where $L \in Q$ because $(N, I)$ accepts $L$. Now every surjective morphism between finite semilattices has the property that the domain has at least as many join-irreducibles as the codomain (since the join-irreducibles form the minimal generating set). Hence we have shown that

$$|J(Q)| \leq |J(\mathcal{P}Z)| = |Z|.$$

(3) Next apply the above construction to the underlying nfa of $\mathcal{N}(L)$. We obtain a JSL-dfa $\mathcal{D}_L = (Q_L, a^{-1}(-), \hat{F})$ where $Q_L$ is the semilattice of languages which this nfa can accept, varying the set of initial states. In fact:

(a) $Q_L$ is precisely the set of unions of derivatives of $L$, see Proposition 3.16 and use that $\mathcal{N}(L)$ is reachable and (isomorphic to) a normalized nca by Corollary 3.12.
(b) Applying $\mathcal{G}_*$ to $(\mathcal{D}_L, L)$ one obtains the minimal pointed nca $\mathcal{N}(L)$. In particular, $Z_L$ has states $J(Q_L)$ – see Lemma 2.19 and Proposition 3.24.

(c) Therefore $CZ_L = CGQ_L \cong Q_L$, since $G$ and $C$ define an equivalence.

It follows from (a) that we have an injective $\text{Jdfa}_*$-morphism:

$$\iota : (\mathcal{D}_L, L) \hookrightarrow (\mathcal{D}', L).$$

Indeed $Q_L \subseteq Q$ because $L \in Q$ and the latter is closed under unions and derivatives. This inclusion certainly preserves unions. The transition structure and final states are inherited, so $\iota$ is well-defined.

(4) Now assume that $\mathcal{N}(L)$ is topological, hence $Q_L \cong CZ_L$ is distributive by Lemma 4.2. In view of (2), it remains to prove that $|J(Q_L)| \leq |J(Q)|$, for which we establish an auxiliary statement: given any finite distributive lattice $D$ which is a sub semilattice of a finite semilattice $S$, we prove that $|J(D)| \leq |J(S)|$. Let $|J(S)| = n$, so we have a surjective join-semilattice homomorphism $\mathcal{P}_n \to S$. By the self-duality of $\text{JSL}_f$ we have an embedding of $S^{op}$ into $(\mathcal{P}_n)^{op} \cong \mathcal{P}_n$. Thus any maximal chain in $S$ (hence also in $D$) has at most $n$ edges. Since the number of join-irreducibles in a finite distributive lattice equals the number of edges of any maximal chain [8, Corollary II.112], it follows that $|J(D)| \leq n = |J(S)|$. Now $Q_L$ is a distributive sub-semilattice of $Q$ by (3), so applying this result to $D = Q_L$ and $S = Q$ proves the theorem.

Example 4.5 The converse of Theorem 4.4 is generally false. Let $L = \overline{aa}$ denote the complement of the singleton $\{aa\}$. The underlying nfa of $\mathcal{N}(L)$ and the lattice $Q_L$ are depicted below:

Clearly $Q_L$ is non-distributive but $\mathcal{N}(L)$ is a state-minimal nfa accepting $L$.

5 Conclusions and Future Work

It has been known since the early days of automata theory that nondeterministic finite automata suffer from two unpleasant phenomena, as opposed to their deterministic counterparts: the lack of canonical machines, and the lack of state-minimization. In this paper, we have demonstrated that both problems disappear when one augments nfas with a closure structure. Based on the equivalence between $\text{JSL}$-dfas and nondeterministic closure automata, we derived the “right” notion of...
minimality, which allowed us to establish the existence of a unique minimal nca for each regular language along with a Brzozowski-style construction method. Furthermore, by restricting to ncas with topological closure, we identified a very natural class of canonical state-minimal nfas.

One open question that we leave for future work is to what extent our main result (Theorem 4.4) can be reversed, that is, under which conditions the state-minimality of \( \mathcal{N}(L) \) implies topologicity.

Another issue we aim to address in the future are the complexity-related implications of our results. Although the general state minimization problem for nfas is known to be \( \text{PSPACE} \)-complete, a good implementation of our operators \( \text{reach} \) and \( \text{rev} \) could lead to efficient state minimization procedures for the class of topological automata, and possibly even larger classes of automata.

References


URL http://doi.acm.org/10.1145/2490818


URL http://books.google.co.uk/books?id=6XJX5-zCoIQC

A Appendix

We prove the claim made in the Introduction that the category $\text{JSL}_f$ of finite semilattices arises as a free cocompletion of $\text{Rel}_f$. More precisely, $\text{JSL}_f$ is both
(a) a free cocompletion of $\text{Rel}_f$ under reflexive coequalizers, and
(b) a conservative cocompletion of $\text{Rel}_f$ under finite colimits.

Let us us first recall the general concepts.

**Definition A.1** Let $\mathcal{A}$ be a category. A **free cocompletion under reflexive coequalizers** of $\mathcal{A}$ is a full embedding $E : \mathcal{A} \to \mathcal{A}'$ such that

(i) $\mathcal{A}'$ has reflexive coequalizers (i.e., coequalizers of pairs of retractions $f, g : X \to Y$ having a common coretraction $d : Y \to X$, $fd = \text{id}_Y = gd$).

(ii) Every functor $F : \mathcal{A} \to \mathcal{B}$, where $\mathcal{B}$ has reflexive coequalizers, has an extension $F' : \mathcal{A}' \to \mathcal{B}$ (i.e., $F'E \cong F$) preserving reflexive coequalizers, which is unique up to natural isomorphism.

**Definition A.2** Let $\mathcal{A}$ be a category with finite coproducts. A **conservative finite cocompletion** of $\mathcal{A}$ is a full embedding $E : \mathcal{A} \to \mathcal{A}'$ such that

(i*) $E$ preserves finite coproducts and $\mathcal{A}'$ is finitely cocomplete.

(ii*) Every finite-coproduct preserving functor $F : \mathcal{A} \to \mathcal{B}$, where $\mathcal{B}$ is finitely cocomplete, has an extension to a finite-colimit preserving functor $F' : \mathcal{A}' \to \mathcal{B}$ (i.e., $F'E \cong F$) which is unique up to natural isomorphism.

**Example A.3** Let $\mathcal{V}$ be a finitary variety, and let $\mathcal{V}_{fp}$ and $\mathcal{V}_{fgf}$ be the full subcategories of all finite presentable and finitely generated free algebras, respectively. Then $\mathcal{V}_{fp}$ is a free cocompletion of $\mathcal{V}_{fgf}$ under reflexive coequalizers, as well as a conservative finite cocompletion of $\mathcal{V}_{fgf}$, see [3, Theorem 7.3 and Theorem 17.11].

**Corollary A.4** $\text{JSL}_f$ is a free cocompletion of $\text{Rel}_f$ under reflexive coequalizers, as well as a conservative finite cocompletion of $\text{Rel}_f$.

**Proof.** Apply the above example to the variety $\mathcal{V} = \text{JSL}$. Here $\mathcal{V}_{fp} = \text{JSL}_f$. Since finitely generated free semilattices are power sets $\mathcal{P}n$ ($n < \omega$), it is easy to see that $\mathcal{V}_{fgf}$ is equivalent to $\text{Rel}_f$. Indeed, the functor $\mathcal{P} : \text{Rel}_f \to \text{JSL}_{fgf}$ assigning to each finite set $n$ its power set and to each relation $h : n \to m$ the semilattice morphism

$$\mathcal{P}h : \mathcal{P}n \to \mathcal{P}m, \quad A \mapsto h[A],$$

is an equivalence of categories. \qed
Coalgebraic Update Lenses

Danel Ahman

Laboratory for Foundations of Computer Science, University of Edinburgh,
10 Crichton Street, Edinburgh EH8 9LE, United Kingdom

Tarmo Uustalu

Institute of Cybernetics at Tallinn University of Technology,
Akadeemia tee 21, 12618 Tallinn, Estonia

Abstract

Lenses are mathematical structures used in the context of bidirectional transformations. In this paper, we introduce update lenses as a refinement of ordinary (asymmetric) lenses in which we distinguish between views and updates. In addition to the set of views, there is a monoid of updates and an action of the monoid on the set of views. Decoupling updates from views allows for other ways of changing the source than just merging a view into the source. We also consider a yet finer dependently typed version of update lenses. We give a number of characterizations of update lenses in terms of bialgebras and coalgebras, including analogs to O’Connor’s coalgebraic and Johnson, Rosebrugh and Wood’s algebraic characterizations of ordinary lenses. We consider conversion of views and updates, a tensor product of update lenses and composition of update lenses.

Keywords: lenses, comonads, monads, coalgebras, algebras, bialgebras, distributive laws, liftings, monoid acts, directed containers

1 Introduction

Lenses are studied in the context of bidirectional transformations. A lens is a structure on two sets mediating actions on them. Think of making sure that two copies of a database on a server and a client computer remain in agreement no matter how the database is changed on the client’s side; this must also work when the client only sees a part of the database, a “view”.

Many different variations of lenses have been considered in the literature. In the seminal work [8], a lens between $X$ and $S$ was defined as a pair of maps $\mathrm{lkp} : X \to S$ and $\mathrm{upd} : X \times S \to X$, subject to appropriate round-trip laws. Here, $X$ is to be
thought of as a set of sources and $S$ as a set of views. O’Connor [18] showed that such lenses are nothing but coalgebras of the array comonad for $S$. Johnson, Rosebrugh and Wood [14] and Gibbons and Johnson [10] characterized lenses as algebras for a certain monad on $\text{Set}/S$.

Other variants of lenses include, for example, quotient lenses [9], symmetric lenses [12], (symmetric) edit lenses [11], and (asymmetric) delta lenses [6].

In this paper, we define a variation of ordinary (asymmetric) lenses, which we call update lenses (this name is justified by their practical motivation and agrees with the terminology of our prior related work [3] on update monads). In short, update lenses differ from ordinary state-based lenses in that they distinguish between views and updates. In addition to the set of views $S$, there is also a monoid $(P, \circ, \oplus)$ of updates and an action $\downarrow$ of $(P, \circ, \oplus)$ on $S$. The update map thus becomes $\text{upd} : X \times P \to X$, allowing for other ways of changing the source than just merging a view into the source. There is also a dependently typed version where every view $s : S$ comes with its own set of updates $P_s$.

We give a number of characterizations of update lenses in terms of bialgebras, algebras and coalgebras, which we derive from standard facts about distributive laws between comonads/functors, compatible composition of comonads, liftings of comonads/functors to categories of coalgebras, distributive laws of monads/functors over comonads/functors, cofree comonads, adjoint monads and comonads etc. [7,5,4,20,21]. We study conversion of views and updates, a tensor of update lenses, composition of update lenses. We also look at a specialization, initializable update lenses. For lack of space, this appears in the appendix.

Of the related work, the closest to ours is that of Hofmann, Pierce and Wagner [11] on (symmetric) edit lenses. They analyze symmetric lenses, and recognize that edits (in our terminology updates) often carry a natural monoidal structure, and are thus best modeled using monoids. As a result of the symmetric setting, edit lenses consist of two monoids, with the $\text{upd}$ operations given by a suitable variation of monoid homomorphisms. A different line of related work is that of Diskin, Xiong, and Czarnecki [6] and Johnson and Rosebrugh [15], on (asymmetric) delta lenses. The latter group of authors also introduced a subclass of delta lenses, called $c$-lenses, with a more succinct definition. Both groups of authors investigate adding more structure to the sets of sources and views, taking them to be reflexive graphs (or, equivalently, categories). The nodes model sources or views, and arrows model deltas (in our terminology updates) between them. The $\text{lkp}$ and $\text{upd}$ operations rely on functoriality to guarantee the correct and composable translation of deltas.

For readability, we develop all our mathematics for $\text{Set}$ (and categories built on $\text{Set}$), but for nearly everything we do it could be replaced by any category with finite products or any Cartesian closed category.
2 Ordinary lenses

Ordinary lenses: the definition

Given a set $S$ (of views), an ordinary (asymmetric) lens for $S$ is a set $X$ (of sources) and maps $\text{lkp} : X \to S$ (viewing a source) and $\text{upd} : X \times S \to X$ (updating a source)\(^3\) satisfying the conditions

$$x = \text{upd}(x, \text{lkp} x) \quad \text{upd}(\text{upd}(x, s), s') = \text{upd}(x, s') \quad \text{lkp}(\text{upd}(x, s)) = s$$

i.e., making the following three diagrams commute

\[
\begin{array}{cccccc}
X & \xrightarrow{\text{id}, \text{lkp}} & (X \times S) \times S & \xrightarrow{\text{upd} \times S} & X \times S & \xrightarrow{\text{upd}} & X \\
\downarrow & & \downarrow & & \downarrow & & \\
X \times S & \xrightarrow{\text{upd}} & X & & & & S \\
\end{array}
\]

A map between two lenses $(X, \text{lkp}, \text{upd})$, $(X', \text{lkp}', \text{upd}')$ is a map $h : X \to X'$ (conversion of a source) satisfying

$$\text{lkp} x = \text{lkp}' (h x) \quad h (\text{upd}(x, s)) = \text{upd}'(h x, s)$$

i.e.,

\[
\begin{array}{cccccc}
X & \xrightarrow{h} & X' & \xrightarrow{\text{lkp}'} & (X \times S) \xrightarrow{\text{h} \times S} & X' \times S \\
\downarrow & & & \downarrow & & \downarrow & & \\
S & \xrightarrow{\text{lkp}} & S & \xrightarrow{\text{upd}} & X & \xrightarrow{h} & X' \\
\end{array}
\]

Lenses for $S$ and maps between them form a category $\text{Lens } S$.

Running example: editing a bookshop database

Consider modelling a database of a small bookshop. We will work with a very simple database, consisting of only one table, containing information about the identity (e.g., book title and author names, or ISBN), price and quantity of every book in the shop. We let $\text{book}$, $\text{price}$, $\text{quantity}$ stand for the sets of allowed values of these data.

The set of database states or sources $X$ is

$$X = \Sigma B \subseteq \text{book}, B \to \text{price} \times \text{quantity}$$

Every such database state consists of a set of books $B$ currently in the database, and an assignment of $\text{price}$ and $\text{quantity}$ to each book in $B$.

A simple example of a set of views $S$ could be

$$S = \Sigma B \subseteq \text{book}, B \to \text{price}$$

\(^3\) We use the letter $X$ for the source set and $S$ for the view set, and the names $\text{lkp}$ for get and $\text{upd}$ for put. This may at first feel confusing, but agrees with what is mnemonic for algebras of update monads [3].
This set of views can be used to define an ordinary lens for editing book prices as follows.

We define $lkp : X \to S$ by simply discarding the quantity field of each book, that is

$$lkp(B, v) = (B, \text{fst} \circ v)$$

We define $upd : X \times S \to X$ by removing all books that are in the database but not in the view, editing the prices and preserving the quantities of all books that are both in the database and the view, and adding all books that are only in the view with quantity 0, that is

$$upd((B, v), (B', v')) = (B', \lambda b. \text{if } b \in B \text{ then } (v' \cdot \text{snd} \cdot v \cdot b) \text{ else } (v' \cdot b, 0))$$

Observe that, for our structure to satisfy the 3rd ordinary lens law, the view argument we pass to $upd$ has to contain all the books we want to leave in the database after the update. Also, as we cannot restrict which books can be edited, we have to allow the lens to be able to add new books to the source, with some default quantity, here 0.

We claim that such forced choices are a real shortcoming of ordinary lenses, and address it in the rest of this paper by developing the theory of update lenses.

Ordinary lenses: some alternative descriptions

Johnson et al. [14] pointed out the category of lenses for $S$ is the same as the category of algebras of the following monad $(T_1, \eta_1, \mu_1)$ on $\text{Set}/S$:

$$T_1(X, lkp) = (X \times S, \text{snd})$$
$$\eta_1(X, lkp) x = (x, lkp x)$$
$$\mu_1((x, s), s') = (x, s')$$

Indeed, an algebra of $(T_1, \eta_1, \mu_1)$ is a map $upd : (X \times S, \text{snd}) \to (X, lkp)$ in $\text{Set}/S$, i.e., a map $upd : X \times S \to X$ satisfying $lkp(upd(x, s)) = s$, which must moreover also satisfy the conditions of an algebra structure of $(T_1, \eta_1, \mu_1)$. The latter amount exactly to $x = upd(x, lkp x)$ and $upd(upd(x, s), s') = upd(x, s')$.

O’Connor [18] observed that the category of lenses is isomorphic to the category of coalgebras of the following comonad $(D, \varepsilon, \delta)$ on $\text{Set}$ (sometimes [19] called the array comonad for $S$):

$$DX = S \times (S \to X)$$
$$\varepsilon(s, v) = vs$$
$$\delta(s, v) = (s, \lambda s'. (s', v))$$

Explicitly, a coalgebra of $(D, \varepsilon, \delta)$ is a set $X$ and a map $act : X \to S \times (S \to X)$ satisfying

$$x = \text{let } (s, v) \leftarrow act x \text{ in } vs$$
$$\text{let } (s, v) \leftarrow act x \text{ in } (s, \lambda s'. (v \cdot s')) = \text{let } (s, v) \leftarrow act x \text{ in } (s, \lambda s'. (s', v))$$

25
Ahman and Uustalu

The isomorphism assigns to a lens structure \((lkp, \text{upd})\) on a set \(X\) the \((D, \varepsilon, \delta)\)-coalgebra structure \(\text{act} = (lkp, \text{cur upd})\) on \(X\).

A further characterization, due to Power and Shkaravska [19], is only an equivalence of categories, not an isomorphism. The category of lenses for \(S\) is equivalent to \(\text{Set}\). With a set \(R\), one associates the lens \((X, lkp, \text{upd})\) defined by \(X = S \times R\), \(lkp(s, r) = s\), \(\text{upd}((s, r), s') = (s', r)\). In the converse direction, a lens \((X, lkp, \text{upd})\) is sent to the set \(R = \{x : X \mid lkp x = s_0\}\) where \(s_0\) is some chosen element of \(S\). The choice of \(s_0\) only makes the difference of an isomorphism: for any \(s, s' : S\), the function \(\lambda x. \text{upd}(x, s') : \{x : X \mid lkp x = s\} \rightarrow \{x : X \mid lkp x = s'\}\) is an isomorphism by the 2nd and 3rd lens conditions. (In the special case \(S = 0\), we take \(R = 0\).)

Intuitively, \(X\) is decomposed into two parts: \(S\), which can be both viewed and updated, and \(R\), which can be neither viewed nor updated.

Composition of ordinary lenses

Rather than thinking of \(S\) (the set of views) as a fixed set, we can let it vary. We can then define that an (ordinary) lens between two sets \(X\) and \(Y\) is a pair of maps \(lkp : X \rightarrow Y\) and \(\text{upd} : X \times Y \rightarrow X\) satisfying the three conditions.

For any set \(X\), there is the identity lens \((\text{id} \{X\}, \text{fst})\). Given two lenses \((lkp, \text{upd}) : X \rightarrow Y\) and \((lkp', \text{upd}') : Y \rightarrow Z\), we define their composition to be \((lkp'', \text{upd}'') : X \rightarrow Z\) where \(lkp'' x = lkp'(lkp x)\) and \(\text{upd}''(x, z) = \text{upd}(x, \text{upd}'(lkp x, z))\). Sets and lenses between them for a category.

3 Update lenses

In this paper, we are interested in a more fine-grained variation of lenses that we call update lenses.

Update lenses: the definition

We define an \textit{act} to be given by a set \(S\) (of views), a monoid \((P, o, \oplus)\) (of updates) and an action \(\downarrow\) of the monoid on the set (describing the outcome of applying any given update on any given view). \(^4\)

An \textit{update lens} for an \textit{act} \((S, (P, o, \oplus), \downarrow)\) is a set \(X\) (of sources) and maps \(lkp : X \rightarrow S\) (viewing a source) and \(\text{upd} : X \times P \rightarrow X\) (updating a source) satisfying the conditions

\[
x = \text{upd}(x, o) \quad \text{upd}(\text{upd}(x, p), p') = \text{upd}(x, p \oplus p') \quad lkp(\text{upd}(x, p)) = lkp x \downarrow p
\]

\(^4\) In the literature, the pair \((S, \downarrow)\) is often called a \((P, o, \oplus)\)-\textit{set}. We will occasionally use this terminology too.
A map between two update lenses \((X, \mathbf{lkp}, \mathbf{upd})\), \((X', \mathbf{lkp}', \mathbf{upd}')\) is a map \(h : X \rightarrow X'\) (conversion of a source) satisfying

\[
\mathbf{lkp} x = \mathbf{lkp}' (h x) \quad \mathbf{h} (\mathbf{upd} (x, p)) = \mathbf{upd}' (h x, p)
\]

i.e.,

\[
X \xrightarrow{\mathbf{h}} X' \quad X \times P \xrightarrow{\mathbf{h} \times P} X' \times P \quad \mathbf{lkp} \downarrow \downarrow \mathbf{lkp}' \quad \mathbf{upd} \downarrow \downarrow \mathbf{upd}'
\]

Running example: editing a bookshop database

We now revisit the bookshop example from Section 2.

We first recall that it was somewhat inconvenient to use the ordinary bookshop lens to edit the price of a particular book. In particular, we could not simply list the books whose prices we wanted to edit, but had to also list all the other books. The main cause for this inconvenience was the single set of views \(S\) that got used in both \(\mathbf{lkp}\) and \(\mathbf{upd}\). Here we illustrate how decoupling the updates from the views helps us reuse the same set of views for different update strategies.

We keep the set of sources \(X\), the set of views \(S\) and the definition of \(\mathbf{lkp}\) as before, but can let \((P, o, \oplus), \downarrow, \mathbf{upd}\) depend on what we want to do with the database.

For the first example, we reconsider editing book prices by taking \(P = (\text{book} \times \text{price})^*\). Any update \(ps : P\) is to be read as a sequence of price modifications of single books. The monoid structure \((o, \oplus)\) is that of free monoids (with \(o = \texttt{[]}\) and \(\oplus = +\)), while \(\downarrow\) and \(\mathbf{upd}\) are defined similarly.

\[
(B, v) \downarrow \texttt{[]} = (B, v)
\]

\[
(B, v) \downarrow (b, p) :: ps = (B, \lambda b'. \text{if } b' = b \text{ then } p \text{ else } v b') \downarrow ps
\]

\[
\mathbf{upd} ((B, v), \texttt{[]}) = (B, v)
\]

\[
\mathbf{upd} ((B, v), (b, p) :: ps) = \mathbf{upd} ((B, \lambda b'. \text{if } b' = b \text{ then } (p, \text{snd} (v b')) \text{ else } v b'), ps)
\]

(With this definition, if a book to modify is not in the database, it is not added!)
Thanks to the freedom offered by update lenses, we can equally well modify the book prices relative to the prices in the database. For this purpose, we can define $P$ to be $(\text{book} \times \text{change})^*$ (let us say a price can only be a nonnegative number, but a change can also be negative). The monoid structure $(P, \circ, \oplus)$ is as above, but the definitions of $\downarrow$ and $\text{upd}^{-1}$ are adjusted.

$$(B, v) \downarrow [] = (B, v)$$

$$(B, v) \downarrow ((b, c) :: ps) = (B, \lambda b'. \text{if } b' = b \text{ then } \max (0, v b' + c) \text{ else } v b') \downarrow ps$$

$\text{upd}^{-1}((B, v), []) = (B, v)$

$\text{upd}^{-1}((B, v), (b, c) :: ps) = \text{upd}^{-1}((B, \lambda b'. \text{if } b' = b \text{ then } (\max (0, \text{fst}(v b') + c), \text{snd}(v b')) \text{ else } v b'), ps)$$

As an additional treat, we can also use the same set of views $S$ to perform both deletions and additions of books. Here we choose $P$ to be $(\text{book} \times \text{price} \times \text{quantity} + \text{book})^*$ where the first summand stands for addition of a book and the second summand for deletion of a book. The monoid structure $(P, \circ, \oplus)$ is again that of free monoids.

$$(B, v) \downarrow [] = (B, v)$$

$$(B, v) \downarrow ((\text{inl}(b, p, q) :: ps) = (B \cup \{b\}, \lambda b'. \text{if } b' = b \text{ then } p \text{ else } v b') \downarrow ps$$

$$(B, v) \downarrow ((\text{inr} b :: ps) = (B \setminus \{b\}, v|_{B \setminus \{b\}}) \downarrow ps$$

$\text{upd}^{-1}((B, v), []) = (B, v)$

$\text{upd}^{-1}((B, v), \text{inl}(b, p, q) :: ps) = \text{upd}^{-1}((B \cup \{b\}, \lambda b'. \text{if } b' = b \text{ then } (p, q) \text{ else } v b'), ps)$

$\text{upd}^{-1}((B, v), \text{inr} b :: ps) = \text{upd}^{-1}((B \setminus \{b\}, v|_{B \setminus \{b\}}), ps)$

Of course, we could also combine addition and deletion of entries with modification of entries, if we wanted.

Notice that changing a book’s price and adding a book are implemented somewhat arbitrarily, if the absolute value of a negative change is greater than the book’s price resp. if the book is already in the database. This is because every update must always be applicable. In Section 7, we will address this deficiency.

**Update lenses as bialgebras of a functor and monad**

The category of lenses for $(S, (P, \circ, \oplus), \downarrow)$ admits several alternative characterizations. We will now consider these in turn. The ultimate insight will be that update lenses are coalgebras of an appropriate comonad. We will arrive there in several small steps through intermediate characterizations that are also of independent value. We only use standard facts about monads, comonads, distributive laws, liftings, cofree comonads, adjoint monads and comonads. These appear scattered in the literature, some references include [7,5,4,20,21].

We begin by noting that a map $\text{lkp}$ from a set $X$ to $S$ is nothing but a coalgebra structure on $X$ of the functor $F_0$ defined by $F_0 X = S$. Also an action $\text{upd}$ of $(P, \circ, \oplus)$
Ahman and Uustalu

on $X$ is exactly an algebra structure on $X$ of the monad $(T_1, \eta_1, \mu_1)$ defined by

$$T_1 X = X \times P$$
$$\eta_1 x = (x, o)$$
$$\mu_1 ((x, p), p') = (x, p \oplus p')$$

Finally, an action $\downarrow : S \times P \to S$ of $(P, o, \oplus)$ on $S$ is exactly a distributive law of the monad $(T_1, \eta_1, \mu_1)$ over the functor $F_0$.

Therefore, an update lens $(X, \text{lkp}, \text{upd})$ for $(S, (P, o, \oplus), \downarrow)$ is nothing but a ↓-bialgebra of the monad $(T_1, \eta_1, \mu_1)$ over the functor $F_0$.

In the same way, update lens maps are bialgebra maps. So the category of update lenses is the same as

1. the category of ↓-bialgebras of the functor $F_0$ and monad $(T_1, \eta_1, \mu_1)$.
2. the category of coalgebras of the ↓-lifting $F_0^\downarrow$ of $F_0$ to the category of algebras of $(T_1, \eta_1, \mu_1)$;
3. the category of algebras of the ↓-lifting $(T_1^\downarrow, \eta_1^\downarrow, \mu_1^\downarrow)$ of $(T_1, \eta_1, \mu_1)$ to the category of coalgebras of $F_0$.

Let us spell out (0') and (0'').

(0'): The category of algebras of $(T_1, \eta_1, \mu_1)$ is $(P, o, \oplus)$-Set. The ↓-lifting $F_0^\downarrow$ of $F_0$ to $(P, o, \oplus)$-Set is defined by $F_0^\downarrow(X, \text{upd}) = (S, \downarrow)$. A coalgebra of $F_0^\downarrow$ is therefore given by a $(P, o, \oplus)$-set $(X, \text{upd})$ with a map $\text{lkp}$ to $(S, \downarrow)$. So the category of coalgebras of $F_0^\downarrow$ is $(P, o, \oplus)$-Set$/S$.

(0''): The category of coalgebras of $F_0$ is Set$/S$. The ↓-lifting $(T_1^\downarrow, \eta_1^\downarrow, \mu_1^\downarrow)$ of $(T_1, \eta_1, \mu_1)$ is defined by

$$T_1^\downarrow(X, \text{lkp}) = (X \times P, \lambda(x, p), \text{lkp} x \downarrow p)$$

An algebra of $(T_1^\downarrow, \eta_1^\downarrow, \mu_1^\downarrow)$ is an object $(X, \text{lkp})$ of Set$/S$ with a map $\text{upd}$ from $(X \times P, \lambda(x, p), \text{lkp} x \downarrow p)$ satisfying the conditions of a monad algebra.

**Update lenses as bialgebras of a comonad and monad**

Let $(D_0, \varepsilon_0, \delta_0)$ be the cofree comonad on the functor $F_0$, i.e., explicitly

$$D_0 X = S \times X$$
$$\varepsilon_0 (s, x) = x$$
$$\delta_0 (s, x) = (s, (s, x))$$

29
Ahman and Uustalu

The category of coalgebras of $F_0$ (i.e., the category $\text{Set}/S$) is isomorphic to the
category of coalgebras of $(D_0, \varepsilon_0, \delta_0)$, whose objects are given by a set $X$ and map
dlkp : $X \to S \times X$ satisfying

$$X \xrightarrow{\text{dlkp}} S \times X \quad X \xrightarrow{\text{dlkp}} S \times X$$

The isomorphism assigns to any $F_0$-coalgebra structure lkp : $X \to S$ on a set $X$ the
$(D_0, \varepsilon_0, \delta_0)$-coalgebra structure

dlkp = $\langle$lkp, id$\rangle$

Furthermore, the distributive law $\downarrow$ of $(T_1, \eta_1, \mu_1)$ over $F_0$ induces a distributive
law $\lambda$ of $(T_1, \eta_1, \mu_1)$ over $(D_0, \varepsilon_0, \delta_0)$, explicitly defined by

$$\lambda : (S \times X) \times P \to S \times (X \times P)$$

As a result, the category of update lenses, i.e, the category of $\downarrow$-bialgebras of $F_0$
and $(T_1, \eta_1, \mu_1)$, is isomorphic to

$$(1) \quad \text{the category of } \lambda \text{-bialgebras of } (D_0, \varepsilon_0, \delta_0) \text{ and } (T_1, \eta_1, \mu_1).$$

Explicitly, the objects of this category are triples of a set $X$, $(D_0, \varepsilon_0, \delta_0)$-coalgebra structure
dlkp : $X \to S \times X$ and $(T_1, \eta_1, \mu_1)$-algebra structure

$$X \times P \xrightarrow{\text{upd}} X \xrightarrow{\text{dlkp}} S \times X$$

This category can also be described as:

$$(1') \quad \text{the category of coalgebras of the } \lambda \text{-lifting } (D_0, \varepsilon_0, \delta_0) \text{ to the category of algebras of } (T_1, \eta_1, \mu_1);$$

$$(1'') \quad \text{the category of algebras of the } \lambda \text{-lifting } (T_1, \eta_1, \mu_1) \text{ to the category of coalgebras of } (D_0, \varepsilon_0, \delta_0).$$

**Update lenses as pairs of coalgebras of a functor and comonad**

Instead of replacing the functor $F_0$ with a comonad, we could replace the monad
$(T_1, \eta_1, \mu_1)$ with a different comonad. This is what we will embark on now.

Consider the following comonad $(D_1, \varepsilon_1, \delta_1)$:

$$D_1 X = P \to X$$

$$\varepsilon_1 v = v \circ$$

$$\delta_1 v = \lambda p \cdot \lambda p' \cdot v(p \oplus p')$$

The functor $D_1$ is the right adjoint of the functor $T_1$. What is more, the comonad
$(D_1, \varepsilon_1, \delta_1)$ is the corresponding “right adjoint” of the monad $(T_1, \eta_1, \mu_1)$ in the
sense of [7]. As a consequence, the category of algebras of \((T_1, \eta_1, \mu_1)\) is isomorphic to the category of coalgebras of \((D_1, \varepsilon_1, \delta_1)\). Explicitly, a coalgebra of \((D_1, \varepsilon_1, \delta_1)\) is an object \(X\) with a map \(\cupd : X \to P \to X\) satisfying

\[
\begin{array}{c}
X \xrightarrow{\cupd} P \to X \\
\ | \downarrow \circ_{\leftarrow} X \\
\downarrow 1 \to X \\
\downarrow \ | \cupd \\
\downarrow \downarrow \ | \cupd \\
X \xrightarrow{\cupd} P \times P \to X \\
\end{array}
\]

The isomorphism assigns to a \((T_1, \eta_1, \mu_1)\)-algebra structure \(\text{upd} : X \times P \to X\) on a set \(X\) the \((D_1, \varepsilon_1, \delta_1)\)-coalgebra structure \(\cupd = \text{cur upd} : X \to P \to X\).

Further the distributive law \(\downarrow\) of \((T_1, \eta_1, \mu_1)\) over \(F_0\) induces a distributive law \(\zeta\) of \(F_0\) over \((D_1, \varepsilon_1, \delta_1)\), namely \(\zeta = \text{cur} \downarrow\). It follows that the category of \(\downarrow\)-bialgebras of \(F_0\) and \((T_1, \eta_1, \mu_1)\) is isomorphic to

\[(2)\] the category of \(\zeta\)-matching pairs of coalgebras of \(F_0\) and \((D_1, \varepsilon_1, \delta_1)\).

Explicitly, the objects of this category are given by triples of a set \(X\), map \(\text{lkp} : X \to S\) and coalgebra structure \(\cupd : X \to P \to X\) of \((D_1, \varepsilon_1, \delta_1)\) satisfying

\[
\begin{array}{c}
S \xleftarrow{\text{lkp}} X \xrightarrow{\cupd = \text{cur upd}} P \to X \\
\ | \downarrow \ P \to \text{lkp} \\
S \xrightarrow{\zeta} P \to S
\end{array}
\]

This category is also describable as

\[(2')\] the category of coalgebras of the lifting \(F_0\) of the functor \(F_0\) to the category of coalgebras of \((D_1, \varepsilon_1, \delta_1)\).

**Update lenses as pairs of coalgebras of two comonads**

We now combine what we did in the last two paragraphs. We replace both \(F_0\) with \((D_0, \varepsilon_0, \delta_0)\) and \((T_1, \eta_1, \mu_1)\) with \((D_1, \varepsilon_1, \delta_1)\). Both the distributive law \(\lambda\) of \((T_1, \eta_1, \mu_1)\) over \((D_0, \varepsilon_0, \delta_0)\) and the distributive law \(\zeta\) of \(F_0\) over \((D_1, \varepsilon_1, \delta_1)\) give us the following distributive law \(\theta\) of \((D_0, \varepsilon_0, \delta_0)\) over \((D_1, \varepsilon_1, \delta_1)\):

\[
\theta : S \times (P \to X) \to P \to S \times X \\
\theta (s, v) = \lambda p. (s \downarrow p, v p)
\]

The category of update lenses, i.e., the category of \(\downarrow\)-bialgebras of the functor \(F_0\) and monad \((T_1, \eta_1, \mu_1)\), is therefore isomorphic to

\[(3)\] the category of \(\theta\)-matching pairs of coalgebras of \((D_0, \varepsilon_0, \delta_0)\) and \((D_1, \varepsilon_1, \delta_1)\).

Explicitly, an object of this category is given by a set \(X\), \((D_0, \varepsilon_0, \delta_0)\)-coalgebra structure \(\text{dlkp} : X \to S \times X\) and \((D_1, \varepsilon_1, \delta_1)\)-coalgebra structure \(\text{cupd} : X \to P \to X\).
Ahman and Uustalu

satisfying

\[
S \times X \xleftarrow{\text{dikp}} S \times (P \to X) \xrightarrow{\text{cupd}} P \to X
\]

\[
S \times S \times (P \to X) \xrightarrow{\theta} P \to X
\]

The same category can also be described as

(3') the category of coalgebras of the \(\theta\)-lifting \((D_0^\theta, \varepsilon_0^\theta, \delta_0^\theta)\) of the comonad \((D_0, \varepsilon_0, \delta_0)\) to the category of coalgebras of \((D_1, \varepsilon_1, \delta_1)\).

**Update lenses as coalgebras of a comonad**

We have two comonads \((D_0, \varepsilon_0, \delta_0)\) and \((D_1, \varepsilon_1, \delta_1)\) with a distributive law \(\theta\) of the former over the latter. This gives a compatible composite comonad \((D, \varepsilon, \delta)\):

\[
DX = S \times (P \to X)
\]

\[
\varepsilon(s, v) = v \circ \delta(s, v) = (s, \lambda p. (s \downarrow p, \lambda p'. v(p \oplus p')))
\]

The category of update lenses, i.e., of \(\theta\)-matching pairs of coalgebras of \((D_0, \varepsilon_0, \delta_0)\) and \((D_1, \varepsilon_1, \delta_1)\), is isomorphic to

(4) the category of \((D, \varepsilon, \delta)\)-coalgebras.

Explicitly, a \((D, \varepsilon, \delta)\)-coalgebra is a set \(X\) with a map \(\text{act} : X \to S \times (P \to X)\) satisfying

\[
x = \text{let } (s, v) \leftarrow \text{act } x \text{ in } v \circ \text{let } (s, v) \leftarrow \text{act } x \text{ in } (s, \lambda p. (s \downarrow p, \lambda p'. v(p \oplus p')))
\]

i.e.,

\[
X \xrightarrow{\text{act}} S \times (P \to X) \xrightarrow{\text{act}} S \times (P \to X)
\]

\[
S \times (P \to X) \xrightarrow{S \times (P \to \text{act})} S \times (P \to S \times (P \to X))
\]

The isomorphism associates to an update lens structure \((\text{lkp}, \text{upd})\) on a set \(X\) a \((D, \varepsilon, \delta)\)-coalgebra structure \(\text{act} = (\text{lkp}, \text{cur upd})\).

Following two routes \((0)-(1)-(2)-(4)\) and \((0)-(1)-(3)-(4)\), we have derived that update lenses for \((S, (P, \circ, \oplus), \downarrow)\) are essentially the same as coalgebras of the comonad \((D, \varepsilon, \delta)\). The algebraic characterization (0'') is analogous to that of Johnson et al. [14] for ordinary lenses for \(S\) and the coalgebraic characterization (4) is analogous to O’Connor’s [18], while perhaps the most basic one is (0'), which says that an update lens is a \((P, \circ, \oplus)\)-set with a map to \((S, \downarrow)\). Many of the other alternative characterizations of update lenses are without counterparts for ordinary lenses.
We have no opportunity to discuss this here in any detail, but update lenses 
(S, (P, o, ⊕), ↓), i.e., coalgbras of the comonad (D, ε, δ), comodel the same (gen-
erally large) Lawvere theory that is modelled by the algebras of what we have 
elsewhere [3] called the update monad for (S, (P, o, ⊕), ↓). This monad (T, η, μ) is 
defined by TX = S → P × X, ηx = λs.(o, x), μf = λs. let \{(p, g) ← f s; (p', x) ← 
g(s ↓ p)\} in (p ⊕ p', x).

### Update lenses: an equivalent characterization

Here is a different characterization, which is only an equivalence of categories, not an isomorphism.

We first observe that any act (S, (P, o, ⊕), ↓) defines a category \( \langle \langle S, (P, o, ⊕), ↓ \rangle \rangle \) 
where an object is an element \( s \) of \( S \) and a map between \( s, s' : S \) is an element \( p \) of \( P \) 
such that \( s ↓ p = s' \). The identity on \( s \) is \( o \) and the composition of \( p \) and \( p' \) is 
\( p ⊕ p' \).

The category \( \text{ULens}(S, (P, o, ⊕), ↓) \) is equivalent to the functor category 
\( \langle \langle S, (P, o, ⊕), ↓ \rangle \rangle, \text{Set} \). A functor \( R : \langle \langle S, (P, o, ⊕), ↓ \rangle \rangle \to \text{Set} \) is mapped to the update lens 
\( (X, \text{lkp}, \text{upd}) \) defined by \( X = \Sigma s : S. R s, \text{lkp}(s, r) = s \) and 
\( \text{upd}((s, r), p) = (s ↓ p, R p r) \). In the converse direction, an update lens 
\( (X, \text{lkp}, \text{upd}) \) is mapped to the functor \( R \) defined by 
\( Rs = \{ x : X \mid \text{lkp} x = s \}, Rp x = \text{upd}(x, p) \).

### Turning ordinary lenses into update lenses

Given an act (S, (P, o, ⊕), ↓), any ordinary lens \( (X, \text{lkp} : X → S, \text{upd} : X × S → X) \) for \( S \) defines an update lens 
\( (X, \text{lkp}, \text{upd}' : X × P → X) \) for (S, (P, o, ⊕), ↓) via 
\( \text{upd}'(x, p) = \text{upd}(x, \text{lkp} x ↓ p) \). And any map \( h \) between two ordinary lenses for \( S \) is 
also a map between the corresponding update lenses for (S, (P, o, ⊕), ↓). This gives 
us a functor from \( \text{Lens} S \) to \( \text{ULens}(S, (P, o, ⊕), ↓) \).

This functor is “caused” by a morphism \( τ \) between the comonads \( (D, ε, δ) \), 
\( (D', ε', δ') \) where \( DX = S × (S → X) \) and \( D'X = S × (P → X) \). It is defined by 
\( τ(s, v) = (s, Ap.v(s ↓ p)) \). A morphism between two comonads gives a functor 
between the corresponding categories of coalgbras.

### 4 Conversions of views and updates

So far we have seen that update lense maps \( h : (X, \text{lkp}, \text{upd}) → (X', \text{lkp}', \text{upd}') \) model 
the conversion from a source set \( X \) to a source set \( X' \), for a fixed act (S, (P, o, ⊕), ↓).
We will now discuss how maps between acts give rise to conversions between view 
and update sets, for a fixed source set \( X \).

A map \((t, q)\) between acts \((S, (P, o, ⊕), ↓)\) and \((S', (P', o', ⊕'), ↓')\) consists of a function 
(conversion of views) \( t : S → S' \) and a monoid homomorphism (conversion 
of updates) \( q : (P', o', ⊕') → (P, o, ⊕) \), such that

\[
t(s ↓ q p) = t s ↓ p
\]

(Notice the reversed direction of the monoid homomorphism! In the literature, e.g.,

Ahman and Uustalu
Ahman and Uustalu

[17], one typically requires that \( q : (P, o, \oplus) \rightarrow (P', o', \oplus') \) and \( t (s \downarrow p) = t s \downarrow' q p \), but this is not what we need here.) Acts form a category \( \text{Act} \).

We say that a conversion of views and updates induced by a morphism \((t, q)\) between acts \((S, (P, o, \oplus), \downarrow)\), \((S', (P', o', \oplus'), \downarrow')\) is a functor

\[
\text{ULens} (t, q) : \text{ULens} (S, (P, o, \oplus), \downarrow) \rightarrow \text{ULens} (S', (P', o', \oplus'), \downarrow')
\]
defined as

\[
\text{ULens} (t, q) (X, \text{lkp}, \text{upd}) = (X, t \circ \text{lkp}, \text{upd} \circ (X \times q))
\]

\[
\text{ULens} (t, q) h = h
\]

\( \text{ULens} \) forms a functor from \( \text{Act} \) to \( \text{CAT} \).

From the functor \( \text{ULens} \) we can build the total category \( \text{ULensTot} \) of all update lenses with the Grothendieck construction (see, e.g., [13, §1.10]), i.e., \( \text{ULensTot} = \int \text{ULens} \).

In detail, an object of \( \text{ULensTot} \) is an act \((S, (P, o, \oplus), \downarrow)\) together with an update lens \((X, \text{lkp}, \text{upd})\) for that act, i.e., an object of \( \text{ULens} (S, (P, o, \oplus), \downarrow) \). A map between two objects \((S, (P, o, \oplus), \downarrow), (X, \text{lkp}, \text{upd})\) and \((S', (P', o', \oplus'), \downarrow'), (X', \text{lkp}', \text{upd}')\) is an act map \((t, q) : (S, (P, o, \oplus), \downarrow) \rightarrow (S', (P', o', \oplus'), \downarrow')\) paired with a map \(h : \text{ULens} (t, q) (X, \text{lkp}, \text{upd}) \rightarrow (X', \text{lkp}', \text{upd}')\) of update lenses for \((S', (P', o', \oplus'), \downarrow')\).

Conversions of views and updates as mappings of comonad coalgebras to comonad coalgebras

Just as every act \((S, (P, o, \oplus), \downarrow)\) defines a comonad \([S, (P, o, \oplus), \downarrow] = (D, \varepsilon, \delta)\), every morphism \((t, q)\) between two acts \((S, (P, o, \oplus), \downarrow)\) and \((S', (P', o', \oplus'), \downarrow')\) defines a morphism \([t, q] = \tau\) between the comonads \([S, (P, o, \oplus), \downarrow]\) and \([S', (P', o', \oplus'), \downarrow']\). Explicitly, this natural transformation is defined by

\[
\tau : \forall \{X\}. (S \times (P \rightarrow X)) \rightarrow S' \times (P' \rightarrow X)
\]

\[
\tau (s, v) = (t s, v \circ q)
\]

\([\cdot]\) forms a functor from \( \text{Act} \) to \( \text{Comonads(Set)} \).

A morphism \( \tau \) between two comonads \((D, \varepsilon, \delta)\) and \((D', \varepsilon', \delta')\) determines a functor between the corresponding categories of coalgebras, sending a \((D, \varepsilon, \delta)\)-coalgebra \((X, \text{act})\) to the \((D', \varepsilon', \delta')\)-coalgebra \((X, \tau \{X\} \circ \text{act})\).

While the category \( \text{ULens} (S, (P, o, \oplus), \downarrow) \) is isomorphic to the category of coalgebras of the comonad \([S, (P, o, \oplus), \downarrow]\), the functor \( \text{ULens} (t, q) : \text{ULens} (S, (P, o, \oplus), \downarrow) \rightarrow \text{ULens} (S', (P', o', \oplus'), \downarrow') \) is isomorphic to the functor between the categories of coalgebras of \([S, (P, o, \oplus), \downarrow]\) and \([S', (P', o', \oplus'), \downarrow']\), induced by the comonad morphism \([t, q]\). In summary, the following diagram commutes up
5 Tensoring update lenses

Sometimes we might want to collect multiple update lenses into a single update lens structure. We show how this can be done with a tensor product on the category of all update lenses.

We define the unit act $I$ to be the act $(1, (1, !, !, !))$.

For any two acts $(S_0, (P_0, o_0, ⊕_0), ↓_0)$ and $(S_1, (P_1, o_1, ⊕_1), ↓_1)$, we define their tensor $(S_0, (P_1, o_1, ⊕_1), ↓_1) ⊗ (S_1, (P_1, o_1, ⊕_1), ↓_1)$ to be $(S_0 \times S_1, (P_0 \times P_1, o, ⊕), ↓)$ where

$$o = (o_0, o_1) \quad (p_0, p_1) ⊕ (p'_0, p'_1) = (p_0 \oplus p'_0, p_1 \oplus p'_1)$$

$I$ and $⊗$ make $\text{Act}$ a monoidal category.

Now for the act $I$ we have a unit lens $J = (1, !, !)$.

And for two update lenses $(X_0, \text{lkp}_0, \text{upd}_0)$ and $(X_1, \text{lkp}_1, \text{upd}_1)$ for acts $(S_0, (P_0, o_0, ⊕_0), ↓_0)$ resp. $(S_1, (P_1, o_1, ⊕_1), ↓_1)$, there is the tensor update lens $(X_0, \text{lkp}_0, \text{upd}_0) ⊗ (X_1, \text{lkp}_1, \text{upd}_1) = (X_0 \times X_1, \text{lkp}, \text{upd})$ for the act $(S_0, (P_0, o_0, ⊕_0), ↓_0) \otimes (S_1, (P_1, o_1, ⊕_1), ↓_1)$ where

$$\text{lkp}(x_0, x_1) = (\text{lkp}_0 x_0, \text{lkp}_1 x_1) \quad \text{upd}(x_0, x_1) (p_0, p_1) = (\text{upd}_0(x_0, p_0), \text{upd}_1(x_1, p_1))$$

With this we have shown that $\text{ULens}$ is a lax monoidal functor from $\text{Act}$ to $\text{CAT}$ (wrt. the $(I, ⊗)$ monoidal structure on $\text{Act}$ and the product monoidal structure on $\text{CAT}$), where the monoidality witnesses are

$$J : 1 \to \text{ULens} I$$

$$\boxtimes : \text{ULens}(S_0, (P_0, o_0, ⊕_0), ↓_0) \times \text{ULens}(S_1, (P_1, o_1, ⊕_1), ↓_1) \to \text{ULens}((S_0, (P_0, o_0, ⊕_0), ↓_0) \otimes (S_1, (P_1, o_1, ⊕_1), ↓_1))$$

It is evident that $(I, ⊗)$ and $(J, \boxtimes)$ also endow the total category $\text{ULensTot}$ with a monoidal structure. The unit is $(I, J)$ and the tensor of $((S_0, (P_0, o_0, ⊕_0), ↓_0), (X_0, \text{lkp}_0, \text{upd}_0))$ and $((S_1, (P_1, o_1, ⊕_1), ↓_1), (X_1, \text{lkp}_1, \text{upd}_1))$ is $((S_0, (P_0, o_0, ⊕_0), ↓_0) \otimes (S_1, (P_1, o_1, ⊕_1), ↓_1)), (X_0, \text{lkp}_0, \text{upd}_0) \boxtimes (X_1, \text{lkp}_1, \text{upd}_1))$.

The tensor product on $\text{ULensTot}$ is a “parallel composition” of lenses.

The category of acts also carries a product monoidal structure. However, due to its involved definition and lack of space, we refrain from discussing it here. For a dependently typed version of acts discussed in Section 7, we worked the details out in the journal version of [2].
6 Composition of update lenses

Above we have seen how to convert of sources (for a fixed act) and views/updates (for a fixed set of sources). We will now discuss a notion of composition of two update lenses where the view set of one update lens is the source set of another.

We develop the definition in two steps, based on the alternative characterization \((0')\) of update lenses from Section 3.

We first recall from \((0')\) that an update lens \((X, \text{lkp} : X \to S, \text{upd} : X \times P \to X)\) for an act \((S, (P, o, \oplus), \downarrow)\) is essentially the same as an object \(((X, \text{upd}), \text{lkp} : (X, \text{upd}) \to (S, \downarrow))\) of \((P, o, \oplus)-\text{Set}/(S, \downarrow)\). If we keep \((P, o, \oplus)\) fixed, but let \((S, \downarrow)\) vary, we can say that an update lens for \((P, o, \oplus)\) is a \((P, o, \oplus)-\text{set map} \text{lkp} : (X, \text{upd}) \to (Y, \text{upd}')\). The identity update lens on \((X, \text{upd})\) is then \(\text{id} \{X\}\) and the composition of two update lenses \(\text{lkp} : (X, \text{upd}) \to (Y, \text{upd}')\) and \(\text{lkp}' : (Y, \text{upd}') \to (Z, \text{upd}'')\) is \(\text{lkp}' \circ \text{lkp}\).

To be able to compose update lenses for possibly different monoids, we need to let \((P, o, \oplus)\) vary too. We achieve this by switching to a more involved category.

An object is just a set. A map between two sets \(X, Y\) is given by a monoid \((P, o, \oplus)\), actions \(\text{upd}\) and \(\downarrow\) on \(X\) resp. \(Y\) (up to isomorphism in the choice of these data) and a map \(\text{lkp} : X \to Y\) satisfying \(\text{lkp}(\text{upd}(x, p)) = \text{lkp}x \downarrow p\).

The identity map on \(X\) is \((P, o, \oplus, \text{upd}, \downarrow, \text{lkp})\) where \(P = X \to X\), \(o = \text{id}\), \(f \oplus g = g \circ f\), \(\text{upd}(x, f) = x \downarrow f = f x\), \(\text{lkp} = \text{id}\). The composition of two maps \((P_0, o_0, \oplus_0, \text{upd}_0, \downarrow_0, \text{lkp}_0) : X \to Y\) and \((P_1, o_1, \oplus_1, \text{upd}_1, \downarrow_1, \text{lkp}_1) : Y \to Z\) is \((P, o, \oplus, \text{upd}, \downarrow, \text{lkp})\) where \(P = \{(p_0, p_1) : P_0 \times P_1 \mid \forall y : Y. y \downarrow_0 p_0 = \text{upd}_1(y, p_1)\}\), \(o = (o_0, o_1)\), \((p_0, p_1) \oplus (p'_0, p'_1) = (p_0 \oplus p'_0, p_1 \oplus p'_1)\), \(\text{upd}(x, (p_0, p_1)) = \text{upd}_0(x, p_0)\), \(z \downarrow (p_0, p_1) = z \downarrow_1 p_1\), \(\text{lkp} = \text{lkp}_1 \circ \text{lkp}_0\).

7 A dependently typed generalization

It often happens that every particular view comes with a specific set of safe or allowed updates. For example, if a view is a table, then one expects to be able to edit and delete only those rows appearing in the view, and not to be able to edit rows after they have been deleted. As we saw in Section 3, update lenses do not capture such intended behavior, albeit it can be accommodated by defining suitably “truncating” \(\downarrow\) and \(\text{upd}\). The reason for this is that there is a single monoid of updates that has to act on every view.

We propose a fix to these limitations of update lenses by introducing a dependently typed version. This development is based on our previous work with Chapman [2] on directed containers, a specialization of Abbott, Altenkirch and Ghani’s containers [1]. Due to lack of space, we must keep the presentation very brief and refer the interested reader to our earlier work [2,3].

Containers, directed containers

Recall that a container is given by a set \(S\) and a \(S\)-indexed family \(P\) of sets. A map between two containers \((S, P)\) and \((S', P')\) is given by functions \(t : S \to S'\) and
Ahman and Uustalu

\[ q : \Pi \{ s : S \}. P' \langle t s \rangle \rightarrow P s. \] Containers and maps between them form a category \textbf{Cont}. Any container \((S, P)\) gives rise to a set functor \([S, P]^c\) defined by

\[
[S, P]^c X = \Sigma s : S. P s \rightarrow X \\
[S, P]^c h = \lambda (s, v). (s, h \circ v)
\]

Any container map \((t, q)\) defines a natural transformation \([t, q]^c : [S, P]^c \rightarrow [S', P']^c\) by \([t, q]^c = \lambda (s, v). (t s, v \circ q)\). \([-]^c\) is a functor from \textbf{Cont} to \([\text{Set}, \text{Set}]\).

A \textit{directed container} \((S, P, \downarrow, \circ, \oplus)\) is given by a container \((S, P)\) and operations

\[
\downarrow : \Pi s : S. P s \rightarrow S \\
\circ : \Pi \{ s : S \}. P s \\
\oplus : \Pi \{ s : S \}. \Pi p : P s. P (s \downarrow p) \rightarrow P s
\]

satisfying five conditions

\[
s \downarrow \circ = s \quad s \downarrow (p \oplus p') = (s \downarrow p) \downarrow p' \\
p \oplus \circ = p \quad \circ \oplus p = p \quad (p \oplus p') \oplus p'' = p \oplus (p' \oplus p'')
\]

Directed containers are a dependently typed generalization of acts. Here we do not have a single monoid and not a family of monoids either, but rather a single monoid-like structure spread out over multiple carriers \(P s\). If one ignores the dependent typing of the above five equations, they are exactly the equations of a monoid and its action on \(S\).

A \textit{map} between directed containers \((S, P, \downarrow, \circ, \oplus)\) and \((S', P', \downarrow', \circ', \oplus')\) is a map \((t, q)\) between the underlying containers \((S, P)\) and \((S', P')\) that satisfies

\[
t \langle s \downarrow q p \rangle = t s \downarrow' p \quad \circ = q \circ' \quad q p \oplus q p' = q (p \oplus' p')
\]

Of course, a map between directed containers is a dependently typed generalization of a map between acts. Directed containers form a category \textbf{DCont}.

A directed container \((S, P, \downarrow, \circ, \oplus)\) determines a comonad \([S, P, \downarrow, \circ, \oplus]^\text{dc} = (D, \varepsilon, \delta)\) given by

\[
D X = [S, P]^c X = \Sigma s : S. P s \rightarrow X \\
\varepsilon (s, v) = v \circ \\
\delta (s, v) = (s, \lambda p. (s \downarrow p, \lambda p'. v (p \oplus p')))
\]

For a map \((t, q)\) between two directed containers \((S, P, \downarrow, \circ, \oplus)\) and \((S', P', \downarrow', \circ', \oplus')\), the natural transformation \([t, q]^c\) is a comonad map between \([S, P, \downarrow, \circ, \oplus]^\text{dc}\) and \([S', P', \downarrow', \circ', \oplus']^\text{dc}\). So the functor \([-]^c : \textbf{Cont} \rightarrow [\text{Set}, \text{Set}]\) lifts to a functor \([-]^\text{dc} : \textbf{DCont} \rightarrow \textbf{Comonads}(\text{Set})\).

In fact, \([-]^\text{dc}\) is the pullback in \textbf{CAT} of \([-]^c\) along \(U : \textbf{Comonads}(\text{Set}) \rightarrow [\text{Set}, \text{Set}]\).
Dependently typed update lenses

A dependently typed update lens \((X, \text{act})\) for a directed container \((S, P, \downarrow, \circ, \oplus)\) is a set \(X\) (of sources) and a map \(\text{act} : X \to \Sigma s : S. Ps \to X\) satisfying

\[
x = \text{let } (s, v) \leftarrow \text{act } x \text{ in } v \circ\n\]
\[
\text{let } (s, v) \leftarrow \text{act } x \text{ in } (s, \lambda p. (s \downarrow p, \lambda p'. v (p \oplus p')))
\]

A map between dependently typed update lenses \((X, \text{act})\) and \((X', \text{act}')\) is a map \(h : X \to X'\) (conversion of a source) satisfying

\[
\text{act}' (h x) = \text{let } (s, v) \leftarrow \text{act } x \text{ in } (s, h \circ v)
\]

The category of dependently typed update lenses is precisely the category of coalgebras of the comonad \((D, \varepsilon, \delta)\) defined above.

While simply-typed update lenses fail to subsume ordinary lenses (since an arbitrary unstructured set \(S\) does not carry monoid structure), an ordinary lens \((X, \text{lkp}, \text{upd})\) for a set \(S\) is a dependently typed updated monad for \((S, P, \downarrow, \circ, \oplus)\) given by \(Ps = S, s \downarrow s' = s', \circ \{s\} = s, s' \oplus \{s\} s'' = s''\).

Running example: editing a bookshop database

We briefly revisit our bookshop example in the context of dependently typed update lenses. We define a dependently typed update lens structure for editing book prices, in a type-safe way.

We keep the set of sources \(X\) and the set of views \(S\) as before. We define the \(S\)-indexed family \(P\) of updates inductively as heterogeneous lists by the two rules

\[
\begin{align*}
\text{} & : P (B, v) \\
v b + c \geq 0 & : P (B, \lambda b' \in B. \text{if } b = b' \text{ then } v' b + c \text{ else } v' b) \\
(b, c) :: & : P (B, v)
\end{align*}
\]

\(P (B, v)\) encodes sequences of single book price changes whose application is guaranteed to lead to no negative price in the database, if all prices in the given view \((B, v)\) of the database are nonnegative.

The monoid structure \((\circ, \oplus)\) is again that of nil and append, but the dependently typed \(\downarrow\) and \(\text{act}\) are defined as below.

\[
\begin{align*}
(B, v) \downarrow \text{} & = (B, v) \\
(B, v) \downarrow ((b, c) :: ps) & = (B, \lambda b' \in B. \text{if } b = b' \text{ then } v b' + c \text{ else } v b') \downarrow ps \\
\text{act} (B, v) & = ((B, \text{fst} \circ v), \lambda ps. \text{act}' (B, v) ps)
\end{align*}
\]

where

\[
\begin{align*}
\text{act}' : & \Pi (B, v) : X. P (B, \text{fst} \circ v) \to X \\
\text{act}' ((B, v), \text{} ) & = (B, v) \\
\text{act}' ((B, v), (b, c) :: ps) & = \\
& \text{act}' ((B, \lambda b'. \text{if } b' = b \text{ then } (\text{fst} (v b') + c, \text{snd} (v b')) \text{ else } v b'), ps)
\end{align*}
\]
8 Conclusions and future work

Combining insights from our work on update monads [3] with existing knowledge about ordinary (asymmetric) lenses [18,14,10], we were led to a concept of update lenses.

Update lenses are a refinement of ordinary lenses that we find both practically meaningful and theoretically elegant. Most interestingly perhaps, update lenses admit decompositions that ordinary lenses do not enjoy: they can be seen as pairs of matching coalgebras, bialgebras etc. These characterizations arise naturally from various kinds of distributive laws.

We wish to continue this work by turning to symmetric variations of update lenses. We expect to be able to build on both the original symmetric lenses works [12,11] and the newest ideas of Johnson and Rosebrugh [16]. We also plan to find out the precise connections to delta lenses and c-lenses [15].

Acknowledgements

Tarmo Uustalu thanks Andreas Abel for reminding him of O’Connor’s work. This ongoing work is being supported by the University of Edinburgh Principal’s Career Development PhD Scholarship, the ERDF funded Estonian CoE project EXCS and ICT National Programme project Coinduction, and the Estonian Research Council target-financed research theme 0140007s12 and grant no. 9475.

References


A Initializable ordinary and update lenses

In this section we discuss a specialization that is applicable to both ordinary and update lenses, initializability.

A.1 Initializable ordinary lenses

In the lenses literature [9,12], one sometimes equips an ordinary lens $(X, \text{lkp}, \text{upd})$ for a set $S$ with an additional map $\text{create} : S \rightarrow X$ (creation of a source from a view) satisfying

$$\text{lkp}(\text{create } s) = s \quad \text{upd}(\text{create } s, s') = \text{create } s'$$

i.e.,

$$S \xrightarrow{\text{create}} X \quad S \times S \xrightarrow{\text{create} \times S} X \times S$$

We call such a structure an initializable lens.
A map between two initializable lenses \((X, \text{lkp}, \text{upd}, \text{create})\) and \((X', \text{lkp}', \text{upd}', \text{create}')\) is a map \(h\) between \((X, \text{lkp}, \text{upd})\) and \((X', \text{lkp}', \text{upd}')\) that also satisfies

\[
h(\text{create} s) = \text{create}' s
\]
i.e.,

\[
\begin{array}{c}
S \\
\text{create} \\
\downarrow
\end{array}
\begin{array}{c}
S \\
\downarrow_{\text{create}'}
\end{array}
\begin{array}{c}
X \\
\rightarrow h
\end{array}
\begin{array}{c}
X'
\end{array}
\]

Of course initializable lenses for \(S\) and maps between them form a category.

### A.2 Initializable update lenses

Update lenses admit this specialization too.

**Initializable update lenses: the definition**

We define an initializable update lens \((X, \text{lkp}, \text{upd}, \text{create})\) for an act \((S, (P, o, \oplus), \downarrow)\) to be given by an update lens \((X, \text{lkp}, \text{upd})\) together with an additional map \(\text{create} : S \rightarrow X\) satisfying

\[
\text{lkp}(\text{create} s) = s \quad \text{upd}(\text{create} s, p) = \text{create} (s \downarrow p)
\]
i.e.,

\[
\begin{array}{c}
S \xrightarrow{\text{create}} X \\
\downarrow_{\text{lkp}} \\
S
\end{array}
\begin{array}{c}
S \times P \xrightarrow{\text{create} \times \text{p}} X \times P \\
\downarrow_{\text{lkp}} \\
S
\end{array}
\begin{array}{c}
\rightarrow \text{upd}
\end{array}
\begin{array}{c}
X
\end{array}
\]

A map between two initializable update lenses \((X, \text{lkp}, \text{upd}, \text{create})\) and \((X', \text{lkp}', \text{upd}', \text{create}')\) is a map \(h\) between \((X, \text{lkp}, \text{upd})\) and \((X', \text{lkp}', \text{upd}')\) that also satisfies

\[
h(\text{create} s) = \text{create}' s
\]
i.e.,

\[
\begin{array}{c}
S \\
\text{create} \\
\downarrow
\end{array}
\begin{array}{c}
S \\
\downarrow_{\text{create}'}
\end{array}
\begin{array}{c}
X \\
\rightarrow h
\end{array}
\begin{array}{c}
X'
\end{array}
\]

Initializable update lenses for \((S, (P, o, \oplus), \downarrow)\) and maps between them form a category.

**Initializable update lenses: alternative descriptions**

Recall the definitions of \(F_0, (T_1, \eta_1, \mu_1)\) and \((D, \varepsilon, \delta)\) from Section 3.
It is immediate from the definition that the category of initializable update lenses is the same as category of triples of a coalgebra of $F_0$, algebra of $(T_1, \eta_1, \mu_1)$ and algebra of $F_0$ on the same carrier pairwise matched by the distributive law $\downarrow$ of $(T_1, \eta_1, \mu_1)$ over $F_0$ (used twice) and the distributive law $\text{id}$ of $F_0$ over itself, i.e., of quadruples of a set $X$, map $\text{lkp} : X \to S$, $(T_1, \eta_1, \mu_1)$-algebra structure $\text{upd} : X \times P \to X$ and map $\text{create} : S \to X$ satisfying

\[
\begin{align*}
X \times P & \xrightarrow{\text{upd}} X \\
& \xrightarrow{\text{lkp}} S \\
S & \xrightarrow{\text{create}} X \\
& \xrightarrow{\text{lkp}} S \\
S \times P & \xrightarrow{\downarrow} S \\
S & \xrightarrow{\text{create}} X \\
& \xrightarrow{\text{upd}} S \\
S & \xrightarrow{\text{create}} S \\
S \times P & \xrightarrow{\text{create}} S
\end{align*}
\]

(remember that a $(T_1, \eta_1, \mu_1)$-algebra structure on $X$ is an action of $(P, o, \oplus)$ on $X$).

We already know that the category of $\downarrow$-matching bialgebras of $F_0$ and $(T_1, \eta_1, \mu_1)$ is isomorphic to the category of coalgebras of $(D, \varepsilon, \delta)$. It is also the case that the distributive laws $\text{id}$ of $F_0$ over itself and $\downarrow$ of $(T_1, \eta_1, \mu_1)$ over $F_0$ make $[\text{id}, \text{cur} \downarrow]$ a distributive law of $F_0$ over $(D, \varepsilon, \delta)$. It follows that the category of initializable update lenses is also isomorphic to the category of bialgebras of $(D, \varepsilon, \delta)$ and $F_0$ matched by the distributive law $(\text{id}, \text{cur} \downarrow)$ of $F_0$ over $(D, \varepsilon, \delta)$, i.e., of triples of a set $X$, $(T_1, \eta_1, \mu_1)$-coalgebra structure $\text{act} : X \to S \times (P \to X)$ and map $\text{create} : S \to X$ satisfying

\[
\begin{align*}
S & \xrightarrow{\text{create}} X \\
& \xrightarrow{\text{act} = (\text{lkp}, \text{cur} \text{upd})} S \times (P \to X) \\
& \xrightarrow{S \times P \to \text{create}} S \times (P \to S)
\end{align*}
\]

But notably we can also pack together $\text{upd}$ and $\text{create}$ instead of $\text{upd}$ and $\text{lkp}$. This is done in two steps.

First we observe that the category of algebras of $F_0$ is isomorphic to the category of algebras of the free monad $(T_0, \eta_0, \mu_0)$ on $F_0$, explicitly defined by

\[
\begin{align*}
T_0 X &= S + X \\
\eta_0 x &= \text{inr} x \\
\mu_0 (\text{inl} s) &= \text{inl} s \\
\mu_0 (\text{inl} (\text{inl} s)) &= \text{inl} s \\
\mu_0 (\text{inl} (\text{inr} x)) &= \text{inr} x
\end{align*}
\]

The isomorphism assigns to a $F_0$-algebra $(X, \text{create})$ the $(T_0, \eta_0, \mu_0)$-algebra $(X, d\text{create})$ where $d\text{create} = [\text{create}, \text{id}]$.

The distributive laws $\text{id}$ of $F_0$ over itself and $\downarrow$ of $(T_1, \eta_1, \mu_1)$ over $F_0$ induce distributive laws $[\text{id}, \text{id}]$ of $(T_0, \eta_0, \mu_0)$ over $F_0$ and $\phi$ of $(T_1, \eta_1, \mu_1)$ over $(T_0, \eta_0, \mu_0)$
Ahman and Uustalu

where

\[
\begin{align*}
\psi : (S + X) \times P & \to S + X \times P \\
\psi (\text{inl } s, p) & = \text{inl} (s \uparrow p) \\
\psi (\text{inr } x, p) & = \text{inr} (x, p)
\end{align*}
\]

The category of initializable update lenses is therefore isomorphic to the category of triples of a coalgebra of \(F_0\), algebra of \((T_1, \eta_1, \mu_1)\) and algebra of \((T_0, \eta_0, \mu_0)\) on a common carrier pairwise matched by the distributive laws \(\downarrow\), \([\text{id}, \text{id}]\) and \(\psi\), i.e., quadruples of a set \(X\), map \(\text{lkp} : X \to S\), \((T_1, \eta_1, \mu_1)\)-algebra structure \(\text{upd} : X \times P \to X\) and \((T_0, \eta_0, \mu_0)\)-algebra structure \(\text{dcreate} : S + X \to X\) satisfying

\[
\begin{array}{ccccccc}
X \times P & \xrightarrow{\text{upd}} & X & \xrightarrow{\text{lkp}} & S & S + X & \xrightarrow{\text{dcreate}} & X \\
\downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow \\
S \times P & \xrightarrow{\downarrow} & S & S + S & \xrightarrow{[\text{id}, \text{id}]} & S & (S + X) \times P & \xrightarrow{\psi} & S + X \times P
\end{array}
\]

Second we notice that the distributive law \(\psi\) of \((T_1, \eta_1, \mu_1)\) over \((T_0, \eta_0, \mu_0)\) defines a compatible composition \((T, \eta, \mu)\) of the two monads, explicitly

\[
\begin{align*}
TX &= S + X \times P \\
\eta x &= \text{inr} (x, 0) \\
\mu (\text{inl} s) &= \text{inl} s \\
\mu (\text{inr} (\text{inl} s, p)) &= \text{inl} (s \uparrow p) \\
\mu (\text{inr} (\text{inr} (x, p), p')) &= \text{inr} (x, p + p')
\end{align*}
\]

The category of algebras of \((T, \eta, \mu)\) is isomorphic to the category of \(\psi\)-matching pairs of algebras of \((T_0, \eta_0, \mu_0)\) and \((T_1, \eta_1, \mu_1)\). The algebra of \((T, \eta, \mu)\) corresponding to a \(\psi\)-matching pair \((X, \text{create}, \text{upd})\) of algebras of \((T_0, \eta_0, \mu_0)\) and \((T_1, \eta_1, \mu_1)\) is \((X, \text{maintain})\) where \(\text{maintain} = [\text{create}, \text{upd}]\).

The distributive laws \(\downarrow\) of \((T_1, \eta_1, \mu_1)\) over \(F_0\) and \([\text{id}, \text{id}]\) of \((T_0, \eta_0, \mu_0)\) over \(F_0\) determine a distributive law \([\text{id}, \downarrow]\) of \((T, \eta, \mu)\) over \(F_0\). As a result, the category of initializable update lenses is isomorphic to the category of bialgebras of \(F_0\) and \((T, \eta, \mu)\) for the distributive law \([\text{id}, \downarrow]\), i.e., triples of a set \(X\), map \(\text{lkp} : X \to S\) and \((T, \eta, \mu)\)-algebra structure \(\text{maintain} : S + X \times P \to X\) satisfying

\[
\begin{array}{cccccc}
S + X \times P & \xrightarrow{\text{maintain}} & X & \xrightarrow{\text{lkp}} & S \\
\downarrow & & \downarrow & & \downarrow \\
S + \text{lkp} \times P & \xrightarrow{[\text{id}, \downarrow]} & S + S \times P & \xrightarrow{\text{lkp}} & S
\end{array}
\]

We see that we can construct initializable update lenses by first giving ourselves the means to initialize and update the source, and only then to view the source.
Abstract

We propose the first sound and complete bisimilarities for the call-by-name and call-by-value untyped $\lambda\mu$-calculus, defined in the applicative style. We give equivalence examples to illustrate how our relations can be used; in particular, we prove David and Py’s counter-example, which cannot be proved with Lassen’s preexisting normal form bisimilarities for the $\lambda\mu$-calculus.

Keywords: contextual equivalence, applicative bisimulation, continuation, control operator

1 Introduction

Contextual equivalence [13] is considered as the most natural behavioral equivalence in languages based on the $\lambda$-calculus. Two terms are equivalent if an outside observer cannot tell them apart when they are evaluated within any context (a term with a hole). However, the quantification over contexts makes proving the equivalence of two given programs cumbersome. Consequently, other characterizations of contextual equivalence are sought for, such as coinductively defined bisimilarities.

Several kinds of bisimilarity have been proposed, such as, e.g., applicative bisimilarity [1], which relates terms by reducing them to values (if possible), and then compares these values by applying them to an arbitrary argument. The idea is the
same for environmental bisimilarity \cite{18}, except the values are tested with arguments built from an environment, which represents the knowledge of an observer about the tested terms. Finally, normal form bisimilarity \cite{11} (initially called open bisimilarity \cite{17}) reduces open terms to normal forms and then compares their subterms. Applicative and environmental bisimilarities still contain some quantification over arguments, and usually coincide with contextual equivalence. In contrast, normal form bisimilarity is easier to use, as its definition does not contain any quantification over arguments, but it is generally not complete, i.e., there exist equivalent terms that are not normal form bisimilar.

This article treats the behavioral theory of the untyped \(\lambda\mu\)-calculus \cite{15}. The \(\lambda\mu\)-calculus provides a computational interpretation of classical natural deduction and thus extends the Curry-Howard correspondence from intuitionistic to classical logic. Operationally, the reduction rules of the calculus express not only function applications but also capturing of the current context of evaluation. Therefore, when considered in the untyped setting, the calculus offers an approach to the semantics of abortive control operators such as \texttt{call/cc} known from the Scheme programming language and it may be viewed as a closely related alternative to Felleisen and Hieb’s syntactic theory of control \cite{6}.

So far no characterization of contextual equivalence has been proposed for either call-by-value or call-by-name \(\lambda\mu\)-calculus. Lassen defined normal form bisimilarities for call-by-name weak-head reduction \cite{10}, for head reduction \cite{12}, and, with Støvring, for call-by-value weak-head reduction \cite{19} that are not complete. However, normal form bisimilarity is complete for the \(\Lambda\mu\)-calculus \cite{5} with head reduction \cite{12}, and also for the \(\lambda\mu\)-calculus with store \cite{19} under call-by-value weak-head reduction. Lassen also defined an incomplete applicative bisimilarity for call-by-name weak-head reduction in \cite{10}. A definition of applicative bisimilarity has also been proposed for a call-by-value typed \(\mu\)PCF \cite{14}, but the resulting relation is neither sound nor complete.

In this work, we propose the first characterizations of contextual equivalence for \(\lambda\mu\)-calculus for both call-by-name and call-by-value weak-head reduction semantics. The applicative bisimilarities we define are harder to use than Lassen’s normal form bisimilarity to prove the equivalence of two given terms, but because they are complete, we can equate terms that cannot be related with normal form bisimilarity, such as David and Py’s counter-example \cite{4}. Even though the two applicative bisimilarities we define are built along the same principles, the relation we obtain in call-by-value is much more difficult to use than the one for call-by-name. However, we provide counter-examples showing that simplifying the call-by-value case so that it matches the call-by-name one leads to an unsound definition.

The paper is organized as follows. We first discuss the behavioral theory of the call-by-name (abbreviated as CBN) \(\lambda\mu\)-calculus in Section 2. We propose a notion of contextual equivalence (in Section 2.2) which observes top-level names, and we then characterize it with an applicative bisimilarity (Section 2.3). In particular, we compare our definition of bisimilarity with Lassen’s work and we prove David and Py’s counter-example using our relation. We then discuss call-by-value (CBV) in
Section 3. We propose a definition of applicative bisimilarity (Section 3.2) which coincides with contextual equivalence. We also provide counter-examples showing that the definition cannot be naively simplified to match the one for call-by-name. Although the relation we obtain is harder to use than the one for call-by-name, we can still prove some interesting equivalences of terms, as we demonstrate in Section 3.3. We conclude in Section 4, and the appendices contain some of the proofs missing from the main text (Appendix A for call-by-name and Appendix B for call-by-value). An accompanying research report [3] also discusses environmental bisimilarity for call-by-name.

2 Call-by-Name $\lambda\mu$-calculus

2.1 Syntax and Semantics

The $\lambda\mu$-calculus [15] extends the $\lambda$-calculus with named terms and a $\mu$ constructor that binds names in terms. We assume a set $X$ of variables, ranged over by $x$, $y$, etc., and a distinct set $A$ of names, ranged over by $a$, $b$, etc. Terms ($T$) and named terms ($U$) are defined by the following grammar:

Terms: 
- $t ::= x$  
- $\lambda x.t$  
- $tt$  
- $\mu a.u$

Named terms: 
- $u ::= [a]t$

Values ($V$), ranged over by $v$, are terms of the form $\lambda x.t$. A $\lambda$-abstraction $\lambda x.t$ binds $x$ in $t$ and a $\mu$-abstraction $\mu a.t$ binds $a$ in $t$. We equate terms up to $\alpha$-conversion of their bound variables and names, and we assume bound names to be pairwise distinct, as well as distinct from free names. We write $fv(t)$ and $fv(u)$ for the set of free variables of, respectively, $t$ and $u$, and we write $fn(t)$ and $fn(u)$ for their set of free names. A term $t$ or named term $u$ is said closed if, respectively, $fv(t) = \emptyset$ or $fv(u) = \emptyset$. Note that a closed (named) term may contain free names. The sets of closed terms, closed values, and named terms are $T^0$, $V^0$, and $U^0$, respectively. In any discussion or proof, we say a variable or a name is fresh if it does not occur in any term under consideration.

We distinguish several kinds of contexts, represented outside-in, as follows:

Contexts: 
- $C ::= \square | C t | t C | \lambda x.C | \mu a.C$

Named contexts: 
- $\square C$

CBN evaluation contexts: 
- $E ::= \square | E t$

Named evaluation contexts: 
- $[a]E$

The syntax of (named) evaluation contexts reflects the chosen reduction strategy, here call-by-name. Contexts can be filled only with a term $t$, to produce either regular terms $C[t]$, $E[t]$, or named terms $\square[t]$, $[a][E[t]]$; the free names and free variables of $t$ may be captured in the process.
We write \( t_0\{t_1/x\} \) and \( u_0\{t_1/x\} \) for the usual capture-avoiding substitution of terms for variables. We define the capture-avoiding substitution of named contexts for names, written \( t(\mathcal{E}/a) \) and \( u(\mathcal{E}/a) \), as follows. Note that the side-condition in the \( \mu \)-binding case can always be fulfilled using \( \alpha \)-conversion.

\[
\begin{align*}
x(\mathcal{E}/a) & \overset{\text{def}}{=} x \\
(\lambda x.t)(\mathcal{E}/a) & \overset{\text{def}}{=} \lambda x.t(\mathcal{E}/a) \\
t_0 t_1(\mathcal{E}/a) & \overset{\text{def}}{=} t_0(\mathcal{E}/a) t_1(\mathcal{E}/a)
\end{align*}
\]

\[
(\mu b.u)(\mathcal{E}/a) \overset{\text{def}}{=} \mu b.u(\mathcal{E}/a) \text{ if } b \notin \text{fn}(\mathcal{E}) \cup \{a\}
\]

We define the CBN reduction relation \( \rightarrow_n \) inductively by the following rules:

\[
\begin{align*}
(\beta_n) & \quad [a](\lambda x.t_0) t_1 \rightarrow_n [a]t_0\{t_1/x\} \\
(\mu) & \quad [a]\mu b.u \rightarrow_n u([a]t/b) \\
(app) & \quad [a]t_0 t_1 \rightarrow_n u([a]t_1/b) \text{ if } [b]t_0 \rightarrow_n u \text{ and } b \notin \text{fn}([a]t_0 t_1)
\end{align*}
\]

Reduction is defined on named terms only. The rule \((\beta_n)\) is the usual call-by-name \( \beta \)-reduction. In rule \((\mu)\), the current continuation, represented by \( a \), is captured and substituted for \( b \) in \( u \). In an application (cf. rule \((app)\)), we reduce the term \( t_0 \) in function position by introducing a fresh name \( b \) which represents the top level. We then replace \( b \) with \([a]t_1 \) in the result \( u \) of the reduction of \([b]t_0 \). We can also express reduction with top-level evaluation contexts as follows.

**Lemma 2.1** \( u \rightarrow_n u' \) iff \( u = \mathcal{E}[(\lambda x.t_0) t_1] \) and \( u' = \mathcal{E}[t_0\{t_1/x\}] \), or \( u = \mathcal{E}[\mu a.u''] \) and \( u' = u''(\mathcal{E}/a) \).

Reduction is also compatible with evaluation contexts in the following sense.

**Lemma 2.2** If \( u \rightarrow_n u' \), then \( u(\mathcal{E}/a) \rightarrow_n u'(\mathcal{E}/a) \).

We write \( \rightarrow_n^* \) for the transitive and reflexive closure of \( \rightarrow_n \), and we define the evaluation relation of the calculus as follows.

**Definition 2.3** We write \( u \downarrow_n u' \) if \( u \rightarrow_n^* u' \) and \( u' \) cannot reduce further.

If \( u \downarrow_n u' \), then \( u' \) is a named value. If \( u \) admits an infinite reduction sequence, we say it diverges, written \( u \uparrow_n \). For example, let \( \Omega \overset{\text{def}}{=} (\lambda x.x) (\lambda x.x) \); then \([a]\Omega \uparrow_n\) for all \( a \).

2.2 Contextual Equivalence

As in the \( \lambda \)-calculus, contextual equivalence in the \( \lambda \mu \)-calculus is defined in terms of convergence. However, unlike previous definitions \([10,12]\), we define contextual equivalence on named terms first, before extending it to any terms.

**Definition 2.4** Two closed terms \( u_0, u_1 \) are contextually equivalent, written \( u_0 \approx_c u_1 \), if for all closed contexts \( C \) and names \( a \), there exist \( b, v_0, \) and \( v_1 \) such that \( C[\mu a.u_0] \downarrow_n [b]v_0 \) iff \( C[\mu a.u_1] \downarrow_n [b]v_1 \).
Note that we can plug only terms in a context, therefore we prefix \( u_0 \) and \( u_1 \) with a \( \mu \)-abstraction. Definition 2.4 is not as generic as it could be, because we require the resulting named values to have the same top-level name \( b \); a more general definition would simply say \( \mathbb{C}[\mu.a.u_0] \Downarrow_n \iff \mathbb{C}[\mu.a.u_1] \Downarrow_n \). Our definition is strictly finer than the general one, because contexts cannot discriminate upon top-level names in some cases, as we can see with the next example.

**Example 2.5** Let \( \Theta \) be Turing’s CBN fixed-point combinator, and let \( v \) be \( \lambda x.\lambda y.(x \times y) \). The terms \( u_0 \) and \( u_1 \) are distinguished by Definition 2.4 if \( a \neq b \), but we show they are related by the general contextual equivalence. To do so, we verify that \( E[\mu.c.u_0] \Downarrow_n \) iff \( E[\mu.c.u_1] \Downarrow_n \) holds for all \( E \) and \( c \), and we can then conclude that \( u_0 \) and \( u_1 \) are in the general equivalence with David and Py’s context lemma [4]. Let \( E \) be of the form \([d]E t\) for some \( d, E, t\). Then \( E[\mu.a.u_0] \Downarrow_n [b]\lambda y.\Theta v \) and \( E[\mu.a.u_1] \Downarrow_n [b]\lambda y.\Theta v \), \( E[\mu.b.u_0] \Downarrow_n [a]\lambda x.\mu.c.E[\lambda y.\Theta v] \) and \( E[\mu.b.u_1] \Downarrow_n [d]\lambda y.\Theta v \), and finally \( E[\mu.c.u_0] \Downarrow_n u_0 \) and \( E[\mu.c.u_1] \Downarrow_n u_1 \) for \( c \notin \{a, b\} \). The case \( E = [d]\Box \) is easy to check as well.

We choose Definition 2.4 because it gives more information on the behaviors of terms than the general equivalence. Besides, only very peculiar terms \( u_0 \) and \( u_1 \) are related by the general equivalence but not by Definition 2.4. These terms are like black holes: they reduce (in some context \( \mathbb{C} \)) to values \([a]v_0 \) and \([b]v_1 \) with \( a \neq b \) that never evaluate their arguments. Indeed, if \( E = [c]\Box t_0 \ldots t_n \), then \( E[\mu.a.v_0] \rightarrow_n E[v_0(\mathbb{E}/a)] \), and \( E[\mu.b.v_1] \Downarrow_n [b]v_1(\mathbb{E}/a) \). Suppose that when evaluating \( E[v_0(\mathbb{E}/a)] \), we evaluate one of the \( t_i \)'s. Then by replacing \( t_i \) with \( \Omega \), we obtain a context \( E' \) such that \( E'[\mu.a.v_0] \uparrow_n \) (because \( \Omega \) will be evaluated), and \( E'[\mu.b.v_1] \Downarrow_n \), which is in contradiction with the fact that \( u_0 \) and \( u_1 \) are in the general equivalence (they are distinguished by \( E'[\mu.a.\mathbb{C}] \)).

We extend Definition 2.4 to any closed terms \( t_0, t_1 \), by saying that \( t_0 \approx_c t_1 \) if \([a]t_0 \approx_c [a]t_1 \) for any fresh \( a \). Other versions of the extension are possible, for example by replacing “for any \( a \)” by “for some \( a \)”, or by dropping the freshness requirement; as can be shown using the results of Section 2.3, all these definitions are equivalent. We can also define contextual equivalence on open terms, using the notion of *open extension*, which extends any relation on closed (named) terms to open (named) terms. We say a substitution \( \sigma \) closes \( t \) (or \( u \)) if \( \sigma \) replaces the variables in \( \text{fv}(t) \) (or \( \text{fv}(u) \)) with closed terms.

**Definition 2.6** Let \( \mathcal{R} \) be a relation on closed (named) terms. Two terms \( t_0 \) and \( t_1 \) are in the open extension of \( \mathcal{R} \), written \( t_0 \mathcal{R}^\circ t_1 \), if for all substitutions \( \sigma \) closing \( t_0 \) and \( t_1 \), we have \( t_0 \sigma \mathcal{R} t_1 \sigma \) (and similarly for \( u_0 \mathcal{R}^\circ u_1 \)).

### 2.3 Applicative Bisimilarity

We propose a notion of applicative bisimulation, which tests values by applying them to a random closed argument. As with contextual equivalence, we give the definitions for named terms, before extending it to regular terms.
Definition 2.7 A relation $R$ on closed named terms is an applicative bisimulation if $u_0 R u_1$ implies

- if $u_0 \to^*_n u_0$, then there exists $u'_1$ such that $u_1 \to^*_n u'_1$ and $u'_0 R u'_1$;
- if $u_0 = [a] \lambda x. t_0$, then there exists $t_1$ such that $u_1 \to^*_n [a] \lambda x. t_1$ and for all $t$, we have $[a] t_0 \langle[a] t/a\rangle t/x \ R [a] t_1 \langle[a] t/a\rangle t/x$;
- the symmetric conditions on $u_1$.

Applicative bisimilarity, written $\approx$, is the largest applicative bisimulation.

For regular terms, we write $t_0 R t_1$ if $[a] t_0 R [a] t_1$ for any $a \notin \text{fn}(t_0, t_1)$. The first item of Definition 2.7 plays the bisimulation game for named terms which are not named values. If $u_0$ is a named value $[a] \lambda x. t_0$, then $u_1$ has to reduce to a named value $[a] \lambda x. t_1$, and we compare the values by applying them to an argument $t$. However, a context cannot interact with $[a] \lambda x. t_0$ and $[a] \lambda x. t_1$ by simply applying them to $t$, because $([a] \lambda x. t_0) t$ is not allowed by the syntax. Consequently, we have to prefix them first with $\mu a$. As a result, we consider the named terms $[a] (\mu a. [a] \lambda x. t_0) t$ and $[a] (\mu a. [a] \lambda x. t_1) t$, which reduce to, respectively, $[a] (\lambda x. t_0 \langle[a] t/a\rangle t)$ and $[a] (\lambda x. t_1 \langle[a] t/a\rangle t)$, and then to $[a] t_0 \langle[a] t/a\rangle t/x$ and $[a] t_1 \langle[a] t/a\rangle t/x$; we obtain the terms in the clause for values of Definition 2.7.

Remark 2.8 When considering $[a] (\mu a. [a] \lambda x. t_0) t$ and $[a] (\mu a. [a] \lambda x. t_1) t$, we use the same top-level name $a$ as the one of the named values $[a] \lambda x. t_0$ and $[a] \lambda x. t_1$. We could use a fresh name $b$ instead; reusing the same name makes the bisimulation proofs easier (we do not have to introduce unnecessary fresh names).

We can also define a big-step version of the bisimulation, where we consider only evaluation to a value.

Definition 2.9 A relation $R$ on closed named terms is a big-step applicative bisimulation if $u_0 R u_1$ implies

- if $u_0 \to^*_n [a] \lambda x. t_0$, then there exists $t_1$ such that $u_1 \to^*_n [a] \lambda x. t_1$ and for all $t$, we have $[a] t_0 \langle[a] t/a\rangle t/x \ R [a] t_1 \langle[a] t/a\rangle t/x$;
- the symmetric condition on $u_1$.

Lemma 2.10 If $R$ is a big-step applicative bisimulation, then $R \subseteq \approx$.

As a first property, we prove that reduction (and therefore, evaluation) is included in bisimilarity.

Lemma 2.11 We have $\to^*_n \subseteq \approx$.

Proof. By showing that $\{(u, u') | u \to^*_n u'\} \cup \{(u, u)\}$ is a big-step bisimulation. □

We give a basic example to show how applicative bisimulation can be used.

Example 2.12 For all closed $v$ and $b \notin \text{fn}(v)$, we prove that $[a] v \approx [a] \lambda x. \mu b. [a] v$ by showing that $\{([a] v, [a] \lambda x. \mu b. [a] v) | b \notin \text{fn}(v)\} \cup \approx$ is an applicative bisimulation. Indeed, if $v = \lambda x. t$, then for all $t'$, we have $[a] t t'/x \approx [a] \mu b. [a] v t'$, because $[a] \mu b. [a] v t' \to^*_n [a] t t'/x$ (and by Lemma 2.11).
2.4 Soundness and Completeness

We now prove that \( \approx \) coincides with \( \approx_c \). We first show that \( \approx \) is a congruence using Howe’s method \([8,7]\), which is a classic proof method to show that an applicative bisimilarity is a congruence. As in \([10]\), we need to slightly adapt the proof to the \( \lambda \mu \)-calculus. Here we only sketch the application of the method, all the details can be found in Appendix A.1.

The principle of the method is to prove that a relation called the Howe’s closure of \( \approx \), which is a congruence by construction, is also a bisimulation. The definition of Howe’s closure relies on an auxiliary relation, called the compatible refinement \( \bar{R} \) of a relation \( R \), and inductively defined by the following rules:

\[
\begin{align*}
x & \bar{R} x \\
\lambda x.t_0 & \bar{R} \lambda x.t_1 \\
\mu a.t_0 & \bar{R} \mu a.t_1 \\
\end{align*}
\]

\[
\begin{align*}
t_0 & \bar{R} t_1 \\
[a]t_0 & \bar{R} [a]t_1 \\
\mathcal{E}_0 & \bar{R} \mathcal{E}_1 \\
\end{align*}
\]

\[
\begin{align*}
t_0 & \bar{R} t_1 \\
\mathcal{E}_0 \, t_0 & \bar{R} \mathcal{E}_1 \, t_1 \\
\mathcal{E}_0 & \bar{R} \mathcal{E}_1 \\
\end{align*}
\]

In the original definition of compatible refinement \([7]\), two terms are related by \( \bar{R} \) if they have the same outer language constructor, and their subterms are related by \( R \). In the \( \lambda \mu \)-calculus, compatible refinement is extended to (named) evaluation contexts, and we allow for the substitution of names with related named contexts.

Given two relations \( R_1 \) and \( R_2 \), we write \( R_1 \circ R_2 \) for their composition, e.g., \( t_0 \bar{R}_1 R_2 t_2 \) holds if there exists \( t_1 \) such that \( t_0 \bar{R}_1 t_1 \) and \( t_1 \bar{R}_2 t_2 \). We can now define Howe’s closure of \( \approx \), written \( \approx^* \), as follows.

**Definition 2.13** The Howe’s closure \( \approx^* \) is the smallest relation verifying:

\[
\approx^0 \subseteq \approx^* \quad \approx^* \approx^0 \subseteq \approx^* \quad \approx^* \subseteq \approx^*
\]

Howe’s closure is defined on open (named) terms as well as on (named) evaluation contexts. Because it contains its compatible refinement, \( \approx^* \) is a congruence. To prove it is a bisimulation, we need a stronger result, called a pseudo-simulation lemma, where we test named values not with the same argument, but with arguments \( t'_0, t'_1 \) related by \( \approx^* \).

**Lemma 2.14** Let \( (\approx^*)^c \) be \( \approx^* \) restricted to closed terms, and let \( u_0 \) \( (\approx^*)^c u_1 \).

- If \( u_0 \rightarrow_n u'_0 \), then \( u_1 \rightarrow_n u'_1 \) and \( u'_0 \) \( (\approx^*)^c u'_1 \).
- If \( u_0 = [a] \lambda x. t_0 \), then \( u_1 \rightarrow_n [a] \lambda x. t_1 \) and for all \( t'_0 \) \( (\approx^*)^c t'_1 \), we have \( [a]t_0 \{[a] t'/a\} \{t'_0/x\} \) \( (\approx^*)^c [a]t_1 \{[a] t'/a\} \{t'_1/x\} \).

With this result, we can prove that \( (\approx^*)^c \) is a bisimulation, and therefore included in \( \approx \). Because it also contains \( \approx \) by definition, we have \( \approx = (\approx^*)^c \), and this
implies that \( \approx \) is a congruence. As a result, \( \approx \) is sound w.r.t. to \( \approx_c \).

**Theorem 2.15** \( \approx \subseteq \approx_c \).

To simplify the proof of completeness (the reverse inclusion), we consider an alternate definition of contextual equivalence, where we test terms with named evaluation contexts only. By doing so, we prove a context lemma in the process.\(^3\)

**Definition 2.16** Let \( u_0, u_1 \) be closed terms. We write \( u_0 \approx_c u_1 \) if for all closed contexts \( E \) and names \( a \), there exist \( b, v_0, v_1 \) such that \( E[\mu a.u_0] \Downarrow_n [b]v_0 \) iff \( E[\mu a.u_1] \Downarrow_n [b]v_1 \).

**Theorem 2.17** \( \approx_c \subseteq \approx_c \subseteq \approx \).

The first inclusion is by definition, and the second one is by showing that \( \approx_c \) is a big-step applicative bisimulation.

### 2.5 Comparison with Lassen’s Work

In [10], Lassen also proposes a definition of applicative bisimilarity that he proves sound, but he conjectures that it is not complete. We discuss here the differences between the two approaches.

Lassen defines a notion of bisimulation for regular terms only, and not for named terms. The definition is as follows.

**Definition 2.18** A relation \( R \) on closed terms is a Lassen applicative bisimulation if \( t_0 \, R \, t_1 \) implies:

- for all \( a \), if \( [a]t_0 \xrightarrow{\cdot}_n [b]\lambda x.t'_0 \), then there exists \( t'_1 \) such that \( [a]t_1 \xrightarrow{\cdot}_n [b]\lambda x.t'_1 \), and for all \( t \), we have \( t'_0 \langle [b]t/\cdot \rangle \{t/x\} \, R \, t'_1 \langle [b]t/\cdot \rangle \{t/x\} \);

- the symmetric condition on \( t_1 \).

Lassen’s definition is quite similar to our definition of big-step applicative bisimulation (Definition 2.9), except it requires \( t_0 \langle [b]t/\cdot \rangle \{t/x\} \, R \, t_1 \langle [b]t/\cdot \rangle \{t/x\} \), which implies that these terms must be related when reduced with any top-level name \( a \). This is more restrictive than our definition, where we compare these terms only with the top-level name \( b \) (or, as discussed in Remark 2.8, we could instead compare \( [c]t_0 \langle [c]t/\cdot \rangle \{t/x\} \) and \( [c]t_1 \langle [c]t/\cdot \rangle \{t/x\} \) for some fresh name \( c \)). To illustrate the difference, we consider Lassen’s counter-example from [10].

**Example 2.19** Let \( t_0 \overset{\text{def}}{=} (\lambda x.\lambda y.x\,x) (\lambda x.\lambda y.x\,x) \), and \( t_1 \overset{\text{def}}{=} \mu a.[a]\lambda y.\mu c.[a]t_0 \) (with \( c \neq a \)). These terms are not bisimilar according to Lassen’s definition. For all \( b \), we have \( [b]t_0 \xrightarrow{\cdot}_n [b]\lambda y.t_0 \) and \( [b]t_1 \xrightarrow{\cdot}_n [b]\lambda y.\mu c.[b]t_0 \). With Lassen’s definition, one has to relate \( t_0 \) and \( \mu c.[b]t_0 \) for any \( t \), which means comparing \( [d]t_0 \) and \( [d]\mu c.[b]t_0 \) for all \( d \). But these two terms are not equivalent if \( d \neq b \).

Lassen conjectures in [10] that these terms are contextually equivalent, and we can indeed prove that they are (big-step) bisimilar with our definition: we just have

---

\(^3\) We cannot directly use David and Py’s context lemma [4], because we use a different notion of contextual equivalence.
to compare \([b]t_0\) and \([b]\mu a\cdot[b]t_0\) (or \([c]t_0\) and \([c]\mu a\cdot[c]t_0\) for some fresh \(c\)) for any \(t\), and both terms evaluate to \([b]\lambda x.t_0\) (or \([c]\lambda x.t_0\)) and are therefore equivalent.

By comparing primarily named terms, as we do in our definition, we can keep track of what happens to the top level, and especially of any connection between the top level and a subterm. In Example 2.19, we can see that it is essential to remember that \(b\) represents the top level in \(\mu c.[b]t_0\), and therefore it does not make sense to compare \([d]t_0\) and \([d]\mu c.[b]t_0\) for any \(d \neq b\), as we have to do with Lassen’s definition. We believe that comparing named terms is essential to obtain completeness w.r.t. contextual equivalence; note that the sound and complete normal form bisimilarity for the \(\lambda \mu \rho\)-calculus [19] is also defined on named terms.

2.6 David and Py’s Counter-Example

In [4], David and Py give a counter-example showing that Böhm’s theorem fails in CBN \(\lambda \mu\)-calculus. They prove that their terms are contextually equivalent using a context lemma. Here we slightly simplify their counter-example, and prove equivalence using applicative bisimilarity. Note that these terms cannot be proved equivalent with (a CBN variant of) eager normal form bisimilarity [10,19].

Example 2.20 Let \(0 \overset{\text{def}}{=} \lambda x.\lambda y.y\), \(1 \overset{\text{def}}{=} \lambda x.\lambda y.x\), and \(t_a \overset{\text{def}}{=} \mu c.[a]0\). Then we have \(\lambda x.\mu a.[a]x \mu b.[a]x t_a 0 \approx \lambda x.\mu a.[a]x \mu b.[a]x t_a 1\)\(^4\).

Proof. [Sketch] We only give the main ideas here, the complete equivalence proof can be found in Appendix A.2. First, \(\lambda x.\mu a.[a]x \mu b.[a]x t_a 0\) is not normal form bisimilar to \(\lambda x.\mu a.[a]x \mu b.[a]x t_a 1\), because the subterms of these two terms are not normal form bisimilar (0 is not equivalent to 1).

To prove applicative bisimilarity, let \(c\) be a fresh name and \(t\) be a closed term. We want to relate \([c]\mu a.[a]t \mu b.[a]tt_a 0\) and \([c]\mu a.[a]t \mu b.[a]tt_a 1\), which reduce respectively to \([c]t \mu b.[c]tt_c 0\) (1) and \([c]t \mu b.[c]tt_c 1\) (2). Let \(d \notin \text{fn}(t)\); we distinguish several cases depending on the behavior of \([d]t\). The interesting case is when \([d]t \Downarrow_n [d]\lambda y.t'\); then \(\mu b.[c]t t_c 0\) or \(\mu b.[c]t t_c 1\) is passed as an argument to \(\lambda y.t'\) in respectively (1) and (2). If \(t'\) executes its argument (that is, if \(t'\) reduces to \(E[y]\) for some \(E\)), then (1) reduces to \([c]t t_c 0\) (3), and (2) to \([c]t t_c 1\) (4). But we know that \([d]t \Downarrow_n [d]\lambda y.t',\) and \(t'\) executes its argument, so when evaluating (3) and (4), \(t_c\) will be reduced, and therefore (3) and (4) will evaluate to \([c]0\).

In the other cases (e.g., \([d]t \Downarrow_n [e]\lambda y.t'\) with \(e \neq d\)), either (1) and (2) eventually get to a point similar to the situation above where \(t_c\) is executed, or they diverge. In all cases, they are applicative bisimilar. □

---

\(^4\) The terms David and Py consider in their work are \(\lambda x.\mu a.[a]x \mu b.[a](x t_a 0)t_a\) and \(\lambda x.\mu a.[a]x \mu b.[a](x t_a 1)t_a\). However, the additional argument \(t_a\) would not come into play in the proof we present, so we have elided it.
3 Call-by-Value $\lambda\mu$-calculus

3.1 Semantics and Contextual Equivalence

In this section, we use CBV left-to-right evaluation, which is encoded in the syntax of the CBV evaluation contexts:

$$E ::= □ | E t | v E$$

The CBV reduction relation $\to_v$ is defined by the following rules.

$$(\beta_v) \quad [a](\lambda x.t) v \to_v [a]t[v/x]$$

$$(\mu) \quad [a]\mu b.u \to_v u([a]□/b)$$

$$(app) \quad [a]t_0 t_1 \to_v u([a]□ t_1/b) \quad \text{if } [b]t_0 \to_v u \text{ and } b \notin \text{fn}([a]t_0 t_1)$$

$$(app_v) \quad [a]v t \to_v u([a]v □/b) \quad \text{if } [b]t \to_v u \text{ and } b \notin \text{fn}([a]v t)$$

With rule $(app_v)$, we reduce arguments to values, to be able to apply CBV $\beta$-reduction (rule $(\beta_v)$). The rules $(\mu)$ and $(app)$ are unchanged. We could also express reduction with top-level named evaluation contexts, as in Lemma 2.1. Furthermore, CBV reduction is compatible with CBV contexts, as in Lemma 2.2. We write $\rightarrow^*_v$ for the reflexive and transitive closure of $\rightarrow_v$, $\Downarrow_v$ for CBV evaluation, and $\Uparrow_v$ for CBV divergence.

We use the same definition of contextual equivalence as in CBN.

Definition 3.1 Let $u_0$, $u_1$ be closed named terms. We write $u_0 \approx_c u_1$, if for all closed contexts $C$ and names $a$, there exist $b$, $v_0$, and $v_1$ such that $C[\mu a.u_0] \Downarrow_v [b]v_0$ iff $C[\mu a.u_1] \Downarrow_v [b]v_1$.

However, unlike in CBN, this definition (where we require the resulting values to have the same top-level names) coincides with the general definition where we simply say “$C[\mu a.u_0] \Downarrow_v$ iff $C[\mu a.u_1] \Downarrow_v$.” Indeed, if $C[\mu a.u_0] \Downarrow_v [b]v_0$ and $C[\mu a.u_1] \Downarrow_v [c]v_1$ with $c \neq b$, then we can easily distinguish them, because $[b]\mu b.C[\mu a.u_0] \Omega \rightarrow^*_v [b]v_0((b)□ □/b) \Omega \Uparrow_v$, and $[b]\mu b.C[\mu a.u_1] \Omega \Downarrow_v [c]v_1((b)□ □/b)$.

We extend $\approx_c$ to any closed terms as in CBN. The definition of open extension is slightly changed in CBV, compared to CBN: we close open terms by substituting their variables with closed values only, and not any closed terms.

3.2 Applicative Bisimilarity

Before giving its complete definition, we explain how applicative bisimilarity $\approx$ should compare two named values $[a]\lambda x.t_0$ and $[a]\lambda x.t_1$. The following reasoning explains and justifies the clauses in Definition 3.4. In particular, we provide counter-examples to show that we cannot simplify this definition.

In CBV $\lambda$-calculus (and also with delimited control [2]), values are tested by applying them to an arbitrary value argument. Following this principle, it is natural to propose the following clause for CBV $\lambda\mu$-calculus.
(1) For all \( v \), we have \([a]_t_0 \langle [a] □ v/a \rangle \{v/x\} \approx [a]_t_1 \langle [a] □ v/a \rangle \{v/x\}\). As with Definition 2.7, we in fact compare \([a] (\mu a. [a]λx.t_0) v\) with \([a] (\mu a. [a]λx.t_1) v\), which reduce to the terms in clause (1). However, such a clause would produce an unsound applicative bisimilarity; it would relate terms that are not contextually equivalent, like the ones in the next example.

**Example 3.2** Let \( v_0 = \lambda x.μb. [a] w x\), \( v_1 = \lambda x. w x x\), and \( w = \lambda y. λz. z y\). Then we have \(([[a]v_0]\langle[|a| □ v/a\rangle = [[a]λx.μb. [a] w x v) v →^*_v [a] w v v v v and \([a]v_1\langle[|a| □ v/a\rangle = [[a]λx. w x x) v →^*_v [a] w v v\). Because they reduce to the same term, \(([[a]v_0]\langle[|a| □ v/a\rangle\) is contextually equivalent to \(([[a]v_1]\langle[|a| □ v/a\rangle\), and using clause (1) would lead us to conclude that \([a]v_0\) and \([a]v_1\) are equivalent as well.

However, \([a]v_0\) and \([a]v_1\) can be distinguished with \( t = μd. [d] λy. μc. [d] w'\), where \( w' = λx. μc. [d] x\ w''\) and \( w'' = λx. λy. λz. Ω\). Indeed, we can check that \(([[a]v_0]\langle[|a| □ t/a\rangle \rightarrow^*_v λz. Ω \text{ and } \([a]v_1\langle[|a| □ t/a\rangle \rightarrow^*_v Ω\). This discrepancy comes from the fact that, in \(([[a]v_1]\langle[|a| □ t/a\rangle\), \( t\) is reduced to a value once, capturing \([a]v_1\ □\) in the process, while \( t\) is reduced twice to a value in \(([[a]v_0]\langle[|a| □ t/a\rangle\), and each time it captures a different context. Therefore, \([a]v_0\) and \([a]v_1\) are distinguished by the context \([a] (μa. □) t\), and they are consequently not contextually equivalent.

Example 3.2 suggests that we should compare \([a]λx.t_0\) and \([a]λx.t_1\) with contexts of the form \([a] □ t\), instead of \([a] □ v\). Therefore, we should compare \( u_0 = [a] (λx.t_0 (\langle[|a| □ t/a\rangle)) t\) with \( u_1 = [a] (λx.t_1 (\langle[|a| □ t/a\rangle)) t\). However, we can restrict a bit the choice of the testing term \( t\), based on its behavior. Let \( b / ∈ fn(t)\); if \([b] t\) diverges, then \( u_0\) and \( u_1\) diverge as well, and we gain no information on \([a]λx.t_0\) and \([a]λx.t_1\) themselves. If \([b] t →^*_v [c] v\) with \( b / c\), then \( u_0 →^*_v [c] v (\langle[|a|λx.t_0 (\langle[|a| □ t/a\rangle) □/b\rangle\), and similarly with \( u_1\). The values \([a]λx.t_0\) and \([a]λx.t_1\) are captured by \([b] t\), and no interaction between \( t\) and the two named values takes place in the process \((\langle[|a|λx.t_0\) and \(\langle[|a|λx.t_1\) are not applied to any value); again, we do not gain any new knowledge on the behavior of \([a]λx.t_0\) and \([a]λx.t_1\). Finally, if \([b] t →^*_v [b] v\), then \( u_0 →^*_v [b] (λx.t_0 (\langle[|a| □ t/a\rangle)) v (\langle[|a|λx.t_0 (\langle[|a| □ t/a\rangle □/b\rangle\), and similarly with \( u_1\); in this case, a value is indeed passed to \(\langle[|a|λx.t_0\) and \(\langle[|a|λx.t_1\), and we can compare their respective behaviors. Therefore, an interaction happens between \( t\) and the tested values iff \( [b] t →^*_v [b] v\), and the results of the interaction (after \( β\)-reduction) are the two terms in the clause below.

(2) For all \( t, b, v\) such that \([b] t →^*_v [b] v\) and \( b / ∈ fn(t)\), we have

\([a]t_0 \langle[a] □ t/a\rangle \{v (\langle[a]λx.t_0 (\langle[a] □ t/a\rangle □/b\rangle) □/x\} \approx [a]t_1 \langle[a] □ t/a\rangle \{v (\langle[a]λx.t_1 (\langle[a] □ t/a\rangle □/b\rangle) □/x\}.

Unfortunately, clause (2) is not enough to obtain a sound bisimilarity. The next example shows that an extra clause is needed.

**Example 3.3** Let \( v_0 = \lambda x. μb. [a] (λy. λz. w y)x\) and \( v_1 = w = \lambda x. w' (x λy. y)\), and \( w' = λy. λz. Ω\). We first show that \([a] v_0\) and \([a] v_1\) are related by clause (2). Let \( t\) such that \([b] t \downarrow v [b] v\) for \( b / ∈ fn(t)\). Then we have \(([a]v_0) \langle[|a| □ t/a\rangle \rightarrow^*_v [a]w' (v (\langle[a]v_0 (\langle[a] □ t/a\rangle □/b\rangle) λx.x)\) and \(([a]v_1) \langle[|a| □ t/a\rangle \rightarrow^*_v [a]w' (v (\langle[a]v_1 (\langle[a] □ t/a\rangle □/b\rangle) λx.x)\). We
can prove that the two resulting terms are contextually equivalent by showing that the relation \( \{ (u[[a]E[v]_0[[a][\square t]/a]] \setminus b), u[[a]E[v][\square/b]] \mid [b]t \downarrow \{ [v], b \notin fn(t) \} \} \) is an applicative bisimulation according to Definition 3.4, and by using Theorem 3.9. The proof can be found in Appendix B.1. Because \(([[a]v]_0)[[a][\square t]/a]\) and \(([[a]v][\square t]/a)\) are contextually equivalent, using only clause (2) would lead us to conclude that \([a]v_0 \text{ and } [a]v_1 \) are also equivalent.

However, these two named values can be distinguished with the context \([a](\lambda x.x x) \mu a.\square\), because in one case we have \(([[a]v]_0)[[a](\lambda x.x x) \square/a] \rightarrow^* \lambda z.\Omega\), and in the other \(([[a]v][[a](\lambda x.x x) \square/a] \rightarrow^* \Omega\). As in Example 3.2, when evaluating \(([a]v_0)[[a](\lambda x.x x) \square/a]\), the body of \(v_0\) is evaluated twice, and two different contexts are captured each time. In contrast, \(v_1\) does not contain any control effect, so when its body is evaluated twice, we get the same result.

Example 3.3 shows that we have to compare two values \([a]\lambda x.t_0\) and \([a]\lambda x.t_1\) by also testing them with contexts of the form \([a]\mu v \square\), i.e., by considering \([a]v \lambda x.t_0[[a][\square v]/a]\) and \([a]v \lambda x.t_1[[a][\square v]/a]\). If \(v = \lambda x.t\), then these terms reduce in one \(\beta\)-reduction step into \([a]t[[a][\lambda x.t_0 [[a][\square v]/a]]/x]\) and \([a]t[[a][\lambda x.t_1[[a][\square v]/a]]/x]\). Taking this and clause (2) into account, we obtain the following definition of applicative bisimulation.

**Definition 3.4** A relation \(R\) on closed named terms is an applicative bisimulation if \(u_0 \rightarrow v u_1\) implies

1. if \(u_0 \rightarrow v u_0\), then there exists \(u_1\) such that \(u_1 \rightarrow^* v u_1\) and \(u_0 \rightarrow^* R u_1\);
2. if \(u_0 = [a]\lambda x.t_0\), then there exists \(t_1\) such that \(u_1 \rightarrow^* v [a]\lambda x.t_1\), and:
   1. for all \(t, b, v\) such that \([b]t \rightarrow^* v [b]v\) and \(b \notin fn(t)\), we have
      \[a]t_0[[a][\square t]/a]\{v[[a][\lambda x.t_0[[a][\square t]/a]]/b]/x\} \rightarrow \[a]t_1[[a][\square t]/a]\{v[[a][\lambda x.t_1[[a][\square t]/a]]/b]/x\};
   2. for all \(v = \lambda x.t\), we have
      \[a]t[[a][\lambda x.t_0[[a][\square v]/a]]/x\} \rightarrow \[a]t[[a][\lambda x.t_1[[a][\square v]/a]]/x\};
3. the symmetric conditions on \(u_1\).

Applicative bisimilarity, written \(\approx\), is the largest applicative bisimulation.

The definition is extended to regular terms \(t_0, t_1\) as in CBN, by using a fresh top-level name \(a\). Note that clause (ii) implies that a bisimulation \(R\) is a congruence w.r.t. (regular) values; indeed, if \(v_0 \rightarrow v_1\), then \([a]v_0 \rightarrow [a]v_1\) for a fresh \(a\), and so we have \([a]t\{v_0/x\} \rightarrow [a]t\{v_1/x\}\) for all \(t\) (by clause (ii)). This property simplifies the congruence proof of \(\approx\) with Howe’s method.

As in CBN, we can define a big-step version of the bisimulation (where we use evaluation instead of reduction), and bisimilarity contains reduction.

**Lemma 3.5** We have \(\rightarrow^* \subseteq \approx\).

The applicative bisimulation for CBV is more difficult to use than the one for CBN, as we can see by considering again the terms of Example 2.12.
Lemma 3.8 The relation pseudo-simulation lemma (similar to Lemma 2.14) beforehand.

Terms (written (≈) and Example 3.2 shows that it is indeed not sufficient.

\[ \lambda \text{be sound in a CBV} \approx \text{result}, \]

contexts (see Definition 2.16), and we prove it is an applicative bisimulation. As a alternate definition of contextual equivalence where we test terms with evaluation.

We can then deduce that (≈) is an applicative bisimulation. As a

result, ≈ coincides with ≈c.

Theorem 3.9 ≈ ≈c.

Remark 3.10 In [9], Koutavas et al. show that applicative bisimilarity cannot be sound in a CBV λ-calculus with exceptions, a mechanism that can be seen as a form of control. Our work agrees with their conclusions, as their definition of applicative bisimilarity compares λ-abstractions by applying them to values only, and Example 3.2 shows that it is indeed not sufficient.
3.3 Examples

Even if applicative bisimulation for CBV is difficult to use, we can still prove some equivalences with it. Here we give some examples inspired from Sabry and Felleisen's axiomatization of call/cc [16]. Given a name \(a\), we write \(a^\dagger\) for the term \(\lambda x.\mu b.[a]x\), and we encode call/cc into \(\lambda x.\mu a.[a]x a^\dagger\). Given a named context \(E\), we also write \(E^\dagger\) for \(\lambda x.\mu b.E[x]\), where \(b \notin \text{fn}(E)\). The first example is the axiom \(C_{\text{tail}}\) of [16], where call/cc is exchanged with a \(\lambda\)-abstraction.

**Example 3.11** If \(y \notin \text{fv}(t_1)\) and \(b\) is fresh, then \([b](\lambda x.\mu a.[a]x a^\dagger) (\lambda y.(\lambda y. t_0)) t_1 \approx [b]((\lambda z.\lambda x.\mu a.[a]x a^\dagger) (\lambda y.t_0)) t_1\).

**Proof.** Let \(v_0 \overset{\text{def}}{=} \lambda z.t_0[b^\dagger/y]\) and \(v_1 \overset{\text{def}}{=} \lambda z.((\lambda x.\mu a.[a]x a^\dagger) (\lambda y.t_0)) t_1\). The term on the left reduces to \([b]v_0 t_1\), so we relate this term to the one the right, i.e., \([b]v_1 t_1\). We distinguish several cases depending on \(t_1\). Let \(c\) be a fresh name. If \([c]t_1 \not\Downarrow_v [c]v\), then \([b]v_0 t_1 \rightarrow [b]v_0 t_0[b^\dagger/y]\{v[\langle b \rangle v_0 □/c]/z\}\) and \([b]v_1 t_1 \rightarrow [b]v_1 t_0[b^\dagger/y]\{v[\langle b \rangle v_1 □/c]/z\}\); because \(c\) is fresh, it does not occur in \(t_0\), and the previous terms can be written \(u[\langle b \rangle v_0 □/c]\) and \(u[\langle b \rangle v_1 □/c]\) with \(u \overset{\text{def}}{=} [b]t_0[b^\dagger/y]\{v/z\}\).

Similarly, if \([c]t_1 \Downarrow_v [d]v\) with \(c \neq d\), then \([b]v_0 t_1 \Downarrow_v [d]v[\langle b \rangle v_0 □/c]\) and \([b]v_1 t_1 \Downarrow_v [d]v[\langle b \rangle v_1 □/c]\). When testing these two values with clauses (i) and (ii), we obtain each time terms of the form \(u[\langle b \rangle v_0 □/c]\) and \(u[\langle b \rangle v_1 □/c]\) for some \(u\). With this reasoning, we can prove that \(\{(u[\langle b \rangle v_0 □/c]), (u[\langle b \rangle v_1 □/c]) | u \in U^0\}\) is an applicative bisimulation, by case analysis on \(u\).

With the next example and congruence of \(\approx\), we can prove the axiom \(C_{\text{abort}}\).

**Example 3.12** Let \(a \neq b\); we have \([b]E[a^\dagger t] \approx [b]a^\dagger t\).

**Proof.** We prove that the relation \(R \overset{\text{def}}{=} \{(u[\langle b \rangle E^\dagger □/c]), (u[\langle b \rangle E^\dagger □/c]) | u \in U^0\}\) is an applicative bisimulation by case analysis on \(u\). For example, if \(u = [c]t\) and \(u \Downarrow_v [c]v\), then \(u[\langle b \rangle E^\dagger □/c]\) \(\rightarrow^* [b]E[\langle b \rangle E^\dagger □/c]\) \(\rightarrow^* [b]E[\langle b \rangle E^\dagger □/c]\) \(\rightarrow^* [b]E[\langle b \rangle E^\dagger □/c]\). We obtain terms of the form \(u'[\langle b \rangle E^\dagger □/c]\) and \(u'[\langle b \rangle E^\dagger □/c]\) that are in \(R\). The remaining cases are similar.

**Example 3.13** [axiom \(C_{\text{ifl}}\)] We have \([b]E[\lambda x.\mu a.[a]x a^\dagger t] \approx [b]E[t (\lambda x.b^\dagger E[x])]\).

**Proof.** In this proof, we use an intermediary result, proved in Appendix B.1: if \(E = E_{\text{fr}}[E]\), then \(E^\dagger \approx \lambda x.E_{\text{fr}}[E[x]]\). The proof of the axiom itself is by case analysis on \(t\). An interesting case is when \([d]t \not\Downarrow_v [d]t'y\) where \(d \notin \text{fn}(t)\). Then \([b]E[\langle \lambda x.\mu a.[a]x a^\dagger t\rangle] \rightarrow^* [b]E[\langle \lambda x.\mu a.[a]x a^\dagger □/d\rangle] \rightarrow^* [b]E[t' \langle \langle \lambda x.\mu a.[a]x a^\dagger □/d\rangle [\langle b \rangle \langle \lambda x.\mu a.[a]x □/d\rangle] E[x]/y] \{\text{with } E = [b]E, \text{ and } [b]E[t (\lambda x.b^\dagger E[x])] \rightarrow^* [b]E[t' \langle \langle \lambda x.b^\dagger E[x]\rangle □/d\rangle [\langle \lambda x.b^\dagger E[x]/y] \{\text{with } E = [b]E, \text{ and } [b]E[t (\lambda x.b^\dagger E[x])] \rightarrow^* [b]E[t' \langle \langle \lambda x.b^\dagger E[x]\rangle □/d\rangle [\langle \lambda x.b^\dagger E[x]/y]]\}. From the intermediary result, and because \(\approx\) is a congruence, we know that \([b]E[t' [\langle b \rangle E^\dagger □/y]] \approx [b]E[t' \langle \lambda x.b^\dagger E[x]/y]]\). Hence, to conclude the proof, one can show that

\[
\{(u_0[\langle b \rangle E[\langle \lambda x.\mu a.[a]x a^\dagger □/d\rangle]], u_1[\langle \langle \lambda x.b^\dagger E[x]\rangle □/d\rangle]) | u_0 \approx u_1, E = E_{\text{fr}}[E]\}\}

57
is an applicative bisimulation. \qed

4 Conclusion

In this work we propose a definition of applicative bisimilarity for CBN and CBV $\lambda\mu$-calculus. Even if the two definitions seem quite different, they follow the same principles. First, we believe it is essential for completeness to hold to relate primarily named terms, and then extend the definition to all terms, as explained when discussing Lassen’s definition of applicative bisimilarity (Section 2.5). The top-level names allow to keep track of how the top level is captured and manipulated in the compared terms.

Then, the idea is to test named values with elementary contexts, $[a]t$ for CBN, and $[a]t$ and $[a]v$ for CBV. In the CBV case, we slightly restrict the terms $t$ tested when considering $[a]t$, but the resulting definition remains complex to use compared to CBN, as we can see with Examples 2.12 and 3.6. However, we provide counter-examples showing that we cannot simplify it further (see Examples 3.2 and 3.3). In CBV as well as in CBN, applicative bisimilarity is harder to use than eager normal form bisimilarity [19], but our relations are complete characterizations of contextual equivalence, and we can therefore prove equivalences of terms that cannot be related with normal form bisimilarity, such as David and Py’s example (see Example 2.20) and Example 3.7. To prove the equivalence between two given $\lambda\mu$-terms, one should start with the bisimulation of [19], and if it fails, try next our applicative (or environmental [3]) bisimulations.

We believe the relations we define remain complete w.r.t. contextual equivalence in other variants of the $\lambda\mu$-calculus (perhaps with some slight variations), such as $\lambda\mu$ with different reduction semantics (like, e.g., in [4]), typed $\lambda\mu$-calculus [15], or de Groote’s extended calculus ($\Lambda\mu$-calculus [5]). However, any direct implications of this work for other calculi for abortive continuations such as the syntactic theory of control [6] are unclear and remain to be investigated. The reason is that our approach hinges on the syntactic notion of names, unique to the $\lambda\mu$-calculus, that allows one to keep track of the whereabouts of the top level.

Acknowledgment

We thank Wojciech Jedynak and the anonymous referees for helpful comments on the presentation of this work. The first author has been supported by the Polish NCN grant number DEC-011/03/B/ST6/00348.

References


A Call-by-Name $\lambda\mu$-calculus

A.1 Soundness and Completeness of Applicative Bisimilarity

Let $(\approx^*)^c$ be the restriction of $\approx^*$ to closed terms.

Lemma A.1 If $t_0 \approx^* t_1$, then there exists a substitution $\sigma$ which closes $t_0$ and $t_1$ such that $t_0 \sigma (\approx^*)^c t_1 \sigma$, and the size of the derivation of $t_0 \sigma (\approx^*)^c t_1 \sigma$ is equal to the size of the derivation of $t_0 \approx^* t_1$. A similar result holds if $u_0 \approx^* u_1$.

Proof. As usual. $\square$

Lemma A.2 Let $t_0 (\approx^*)^c t_1$, and $a \notin \text{fn}(t_0) \cup \text{fn}(t_1)$.
Proof. By induction on \( t_0 \) \( \approx^\ast \) \( t \).

Suppose \( t_0 \approx^\ast t \approx^\circ t_1 \), so that \( t \) is closed (using Lemma A.1 if necessary). In fact, we have \( t_0 \approx^\ast t \approx t_1 \). If \([a]t_0 \rightarrow_n u_0\), then by the induction hypothesis, there exists \( u \) such that \([a]t \rightarrow^*_n u \) and \( u_0 \approx^\ast u \). By bisimilarity, there exists \( u_1 \) such that \([a]t_1 \rightarrow^*_n u_1 \) and \( u \approx u_1 \). From \( u_0 \approx^\ast u \approx u_1 \), we deduce \( u_0 \approx^\ast u_1 \), as wished.

Suppose \( t_0 = \lambda x.t'_0 \) and let \( t''_0 \approx^\ast t''_1 \). By induction, there exists \( t' \) such that \( [a]t \rightarrow^*_n [a]\lambda x.t' \) and \([a]t'_0 \{t''_0/x\} \approx^\ast [a]t'_1 \{t''_1/x\} \). By bisimilarity, there exists \( t'_1 \) such that \([a]t_1 \rightarrow^*_n [a]\lambda x.t'_1 \) and \([a]t'_0 \{t''_0/x\} \approx [a]t'_1 \{t''_1/x\} \). Hence, we have \([a]t'_0 \{t''_0/x\} \approx^\ast [a]t'_1 \{t''_1/x\} \), i.e., \([a]t'_0 \{t''_0/x\} \approx^\ast [a]t'_1 \{t''_1/x\} \) as required.

If \( t_0 \) \( \approx^\ast \) \( t \), then we have several cases to consider.

Suppose \( t_0 = \lambda x.t'_0 \) and \( t_1 = \lambda x.t'_1 \) with \( t'_0 \approx^\ast t'_1 \). Let \( t''_0 \approx^\ast t''_1 \). We have \([a]t'_0 \{t''_0/x\} \approx^\ast [a]t'_1 \{t''_1/x\} \), hence the result holds (note that \( a \notin \text{fv}(t'_1) \) because \( a \notin \text{fv}(t_1) \)).

Suppose \( t_0 = t_0 \) and \( t_1 = t_1 \) with \( t'_0 \approx^\ast t'_1 \). We distinguish two cases.

- If \([b]t'_0 \rightarrow_n u_0 \) (for some fresh \( b \)), then \([a]t_0 \rightarrow_n [a][\lambda x.t'_0] \). By the induction hypothesis, there exists \( u_1 \) such that \([b]t'_1 \rightarrow^*_n u_1 \) and \([a]u_0 \approx^\ast u_1 \). Consequently, we have \([a]t_1 \rightarrow^*_n [a][\lambda x.t'_1] \), and by definition of \( \approx^\ast \), we have \([a]u_0 \approx^\ast u_1 \), as required.

- If \( t'_0 = \lambda x.t'_0 \), then we have \([a]t_0 \rightarrow_n [a]t'_0 \{t''_0/x\} \). By the induction hypothesis, there exists \( t'_1 \) such that \([a]t'_1 \rightarrow^*_n [a][\lambda x.t'_1] \) and \([a]t'_0 \{t''_0/x\} \approx^\ast [a]t'_1 \{t''_1/x\} \). From \([a]t'_1 \rightarrow^*_n [a][\lambda x.t'_1] \), we deduce

\[
[a]t_1 \rightarrow^*_n [a][\lambda x.t'_1(\langle a \square t''_0/x \rangle)]t'_1 \rightarrow_n [a]t'_1(\langle a \square t''_0/x \rangle),
\]

hence the result holds.
Suppose \( t_0 = t_0' (\varepsilon_0 / b) \), \( t_1 = t_1' (\varepsilon_1 / b) \) with \( t_0' (\cong \bullet) t_1' \), \( \varepsilon_0 (\cong \bullet) \varepsilon_1 \). If \([a]t_0 \rightarrow_n u_0\), then in fact \([a]t_0' \rightarrow_n u_0'\) with \( u_0 = u_0' (\varepsilon_0 / b)\). By the induction hypothesis, there exists \( u_1'\) such that \([a]t_1' \rightarrow_n u_1'\) and \( u_0' (\varepsilon_0 / b) (\cong \bullet) u_1'\). Consequently, we have \([a]t_1 \rightarrow_n u_1' (\varepsilon_1 / b)\), and \( u_0 (\varepsilon_0 / b) (\cong \bullet) u_1'\) holds, as wished. If \([a]t_0\) is a named value, then in fact \( t_0' = \lambda x.t_1 m\) and \( t_0 = \lambda x.t_1 n\). Let \( t_0' (\cong \bullet) t_1'\). Then \( t_0' (\cong \bullet) t_1'\) for a fresh \( c\). By the induction hypothesis, there exists \( t_1''\) such that \([a]t_1' \rightarrow_n [a]\lambda x.t_1''\) and
\[
[a]t_1'' (t_0'' (\langle c \Box / b \rangle / x)) (\cong \bullet) [a]t_1'' (t_0'' (\langle a \Box (\langle c \Box / b \rangle / a) \rangle \langle t_1'' (\langle c \Box / b \rangle / x) \rangle).
\]
Therefore, we have \([a]t_1 \rightarrow_n [a]\lambda x.t_1'' (\varepsilon_1 / b)\), and
\[
[a]t_1'' (t_0'' (\langle c \Box / b \rangle / x)) (\cong \bullet) [a]t_1'' (t_0'' (\langle a \Box (\langle c \Box / b \rangle / a) \rangle \langle t_1'' (\langle c \Box / b \rangle / x) \rangle (\varepsilon_1 / b))\).
\]
Because \( b\) does not occur in \( t_0'' (\langle c \Box / b \rangle)\), \( t_1'' (\langle c \Box / b \rangle)\) thanks to the renaming to a fresh \( c\), we can switch the substitutions around, and in fact
\[
[a]t_1'' (\varepsilon_0 / b) (t_0'' (\langle c \Box / b \rangle / x)) (\cong \bullet) [a]t_1'' (\varepsilon_1 / b) (t_0'' (\langle a \Box (\langle c \Box / b \rangle / a) \rangle \langle t_1'' (\langle c \Box / b \rangle / x) \rangle)
\]
holds. Renaming \( c\) back into \( b\), we obtain
\[
[a]t_1'' (\varepsilon_0 / b) (t_0'' / x) (\cong \bullet) [a]t_1'' (\varepsilon_1 / b) (t_0'' (\langle a \Box (\langle c \Box / b \rangle / a) \rangle \langle t_1'' / x \rangle)
\]
which gives us the required result.

\[
\square
\]

**Lemma A.3** Let \( u_0 (\cong \bullet) u_1\).

- If \( u_0 \rightarrow_n u_0'\), then \( u_1 \rightarrow_n u_1'\) and \( u_0' (\cong \bullet) u_1'\).
- If \( u_0 = [a]\lambda x.t_0\), then \( u_1 \rightarrow_n [a]\lambda x.t_1\) and for all \( t_0' (\cong \bullet) t_1'\), we have
  \[
  [a]t_0 (\langle a \Box t_0' / a \rangle \langle t_0' / x \rangle) (\cong \bullet) [a]t_1 (\langle a \Box t_1' / a \rangle \langle t_1' / x \rangle).
  \]

**Proof.** By induction on \( u_0 (\cong \bullet) u_1\).

Suppose \( u_0 \approx u_1\). The first item holds by bisimilarity. Suppose \( u_0 = [a]\lambda x.t_0\), and let \( t_0' (\cong \bullet) t_1'\). By definition of the bisimilarity, we have \( u_1 \rightarrow_n [a]\lambda x.t_1\) and \( [a]t_0 (\langle a \Box t_0' / a \rangle \langle t_0' / x \rangle) \approx [a]t_1 (\langle a \Box t_1' / a \rangle \langle t_1' / x \rangle)\). From \( t_0' (\cong \bullet) t_1'\), we deduce \( t_0 (\langle a \Box t_0' / a \rangle \langle t_0' / x \rangle) (\cong \bullet) t_0 (\langle a \Box t_1' / a \rangle \langle t_1' / x \rangle)\), which implies
\[
t_0 (\langle a \Box t_0' / a \rangle \langle t_0' / x \rangle) (\cong \bullet) [a]t_1 (\langle a \Box t_1' / a \rangle \langle t_1' / x \rangle),
\]
which in turn implies
\[
[a]t_0 (\langle a \Box t_0' / a \rangle \langle t_0' / x \rangle) (\cong \bullet) [a]t_1 (\langle a \Box t_1' / a \rangle \langle t_1' / x \rangle),
\]
as required.

Suppose \( u_0 \approx \bullet u \approx \circ u_1\), so that \( u\) is closed (using Lemma A.1 if necessary). In fact, we have \( u_0 (\cong \bullet) u \approx u_1\). If \( u_0 \rightarrow_n u_0'\), then by the induction hypothesis, there
exists \( u' \) such that \( u \rightarrow^* u' \) and \( u_0 \) (\( \approx \)) to \( u' \). By bisimilarity, there exists \( u'_1 \) such that \( u_n \rightarrow^* u'_1 \) and \( u' \approx u'_1 \). From \( u'_0 \) (\( \approx \)) to \( u'_1 \), we deduce \( u'_0 \) (\( \approx \)) to \( u'_1 \), as wished.

Suppose \( u_0 = [a]t_0 \) and let \( t'_0 \) (\( \approx \)) to \( t'_1 \). By induction, there exists \( t \) such that \( u \rightarrow^*_n [a]t_0 \) and \( [a]t_0 \) (\( \approx \) to \( [a]t_0 \) (\( \approx \)) \( t'_1 \)). By bisimilarity, there exists \( t_1 \) such that \( u_1 \rightarrow^*_n [a]t_1 \) and \( [a]t_1 \) (\( \approx \)) to \( [a]t_1 \) (\( \approx \)) \( t'_1 \). Consequently, we have

\[
[a]t_0 \langle [a]t'_0 \rangle \langle t'_1 \rangle (\approx) [a]t \langle [a]t'_1 \rangle \langle t'_1 \rangle,
\]

i.e., \([a]t_0 \langle [a]t'_0 \rangle \langle t'_1 \rangle (\approx) [a]t \langle [a]t'_1 \rangle \langle t'_1 \rangle\) holds as required.

If \( u_0 \) (\( \approx \)) to \( u_1 \), then we have two cases.

Suppose \( u_0 = [a]t_0 \) and \( u_1 = [a]t_1 \) with \( t_0 \) (\( \approx \)) to \( t_1 \). If \( a \notin fn(t_0) \cup fn(t_1) \), then we can apply Lemma A.2 directly to get the required result. Otherwise, let \( b \notin fn(t_0) \cup fn(t_1) \). If \( u_0 \rightarrow^*_n u_0' \) and \( u_0' = [a]t_0 \) (\( \approx \)) to \( [a]t_1 \). We can apply Lemma A.2 to \([b]t_0 \) and \([b]t_1 \), and then rename \( b \) into \( a \).

Suppose \( u_0 = u'_0(\mathbb{E}_0/a), u_1 = u'_1(\mathbb{E}_1/a) \) with \( u'_0 \) (\( \approx \)) to \( u'_1 \) and \( \mathbb{E}_0 \) (\( \approx \)) to \( \mathbb{E}_1 \).

If \( u_0 \) is a named value, then we distinguish two cases. First, we may have \( u'_0 = [a]t_0 \) and \( \mathbb{E}_0 = \mathbb{E}_1 = [b]t_0 \), and \( u_0 = [b]t_0 \) (\( \approx \)) to \( t_1 \). Let \( t_0' \) (\( \approx \)) to \( t'_1 \). Let \( t_0'' \) and \( t_1'' \) be \( t_0' \) and \( t_1' \) where \( a, b \) are renamed into fresh \( c, d \) to avoid some name clashes (we still have \( t_0'' \) (\( \approx \)) to \( t_1'' \)). By the induction hypothesis, there exists \( t_1 \) such that \( u'_1 \rightarrow^*_n [a]t_1 \) and \( [a]t_0 \) (\( \approx \)) to \( [a]t_1 \) (\( \approx \)) \( t_0'' \). This implies

\[
[a]t_0 \langle [a]t'_0 \rangle \langle a \rangle \langle t'_1 \rangle (\approx) [a]t \langle [a]t'_1 \rangle \langle a \rangle \langle t'_1 \rangle,
\]

which is the same as

\[
[a]t_0 \langle [a]t'_0 \rangle \langle a \rangle \langle t'_1 \rangle (\approx) [a]t_1 \langle [a]t'_1 \rangle \langle a \rangle \langle t'_1 \rangle
\]

because \( b \) does not occur in \( t_0'' \), \( t_1'' \). In turn, we have

\[
[b]t_0 \langle [b]t'_0 \rangle \langle a \rangle \langle t'_1 \rangle (\approx) [b]t_1 \langle [b]t'_1 \rangle \langle a \rangle \langle t'_1 \rangle
\]

which is equal to

\[
[b]t_0 \langle [b]t'_0 \rangle \langle a \rangle \langle t'_1 \rangle (\approx) [b]t_1 \langle [b]t'_1 \rangle \langle a \rangle \langle t'_1 \rangle
\]

because \( a \) does not occur in \( t_0'', t_1'' \). Renaming \( c, d \) back into \( a, b \), we obtain

\[
[b]t_0 \langle [b]t'_0 \rangle \langle a \rangle \langle t'_1 \rangle (\approx) [b]t_1 \langle [b]t'_1 \rangle \langle a \rangle \langle t'_1 \rangle
\]

and because \( u_1 \rightarrow^*_n [b]t_1 \), the result holds.
In the second case, we have \( u'_0 = [b] \lambda x.t_0 \) and \( u_0 = [b] \lambda x.t_0.\langle \exists_0/a \rangle \) with \( b \neq a \). Let \( t'_0 \ (\approx^*)c t'_1 \). Let \( t'_0 \), \( t'_1 \) be \( t'_0 \) and \( t'_1 \) where \( a \) is renamed into a fresh \( c \). By the induction hypothesis, there exists \( t_1 \) such that \( u'_1 \rightarrow_n^* [b] \lambda x.t_1 \) and

\[
[b]t_0 (\langle b \rangle \square b' / b) \{t'_0 / x\} \ (\approx^*) c [b]t_1 (\langle b \rangle \square b) \{t'_1 / x\}.
\]

From \( \exists_0 (\langle b \rangle \square b' / b) \ (\approx^*) c \ E_1 (\langle b \rangle \square b) / a' / b \) and the previous relation, we can deduce

\[
[b]t_0 (\langle b \rangle \square b' / b) \{t'_0 / x\} \ (\approx^*) c [b]t_1 (\langle b \rangle \square b) \{t'_1 / x\}.
\]

Because \( a \) does not occur in \( t'_0, t'_1 \), this can be rewritten into

\[
[b]t_0 \langle \exists_0 / a \rangle \langle b \rangle \square b' / b \{t'_0 / x\} \ (\approx^*) c [b]t_1 \langle \exists_1 / a \rangle \langle b \rangle \square b' / b \{t'_1 / x\}.
\]

By renaming \( c \) back into \( a \), we obtain

\[
[b]t_0 \langle \exists_0 / a \rangle \langle b \rangle \square b' / b \{t'_0 / x\} \ (\approx^*) c [b]t_1 \langle \exists_1 / a \rangle \langle b \rangle \square b' / b \{t'_1 / x\},
\]

and because \( u_1 \rightarrow_n^* [b] \lambda x.t_1.\langle \exists_1 / a \rangle \), the result holds.

If \( u_0 \rightarrow_n \), then again we distinguish two cases. First, suppose \( u'_0 \rightarrow_n u''_0 \); then \( u_0 \rightarrow_n u''_0.\langle \exists_0 / a \rangle \). By the induction hypothesis, there exists \( u''_0 \) such that \( u'_1 \rightarrow_n u''_1 \) and \( u''_0 \ (\approx^*) c u''_1 \). Then \( u_1 \rightarrow_n u''_1.\langle \exists_1 / a \rangle \) and \( u''_0.\langle \exists_0 / a \rangle \ (\approx^*) c u''_1.\langle \exists_1 / a \rangle \), hence the result holds.

Otherwise, \( u'_0 = [a] \lambda x.t_0 \), \( E_0 = E'_0 [\exists_0] \), and \( u_0 \rightarrow_n E'_0 [t_0.\langle \exists_0 / a \rangle \{t'_0 / x\}] \). We can prove by induction on \( E_0 \ (\approx^*) c E_1 \) that \( E_1 = E'_1 [\exists_1] \) with \( E'_0 \ (\approx^*) c E'_1 \) and \( t'_0 \ (\approx^*) c t'_1 \). Let \( E'_0, t'_0 \) be \( E'_1, t'_1 \) with \( a \) renamed into a fresh \( b \). By the induction hypothesis, there exists \( t_1 \) such that \( u'_1 \rightarrow_n^* [a] \lambda x.t_1 \), and

\[
[a]t_0 \langle [a] \square b' / a \rangle \{t'_0 / x\} \ (\approx^*) c [a]t_1 \langle [a] \square b' / a \rangle \{t'_1 / x\}.
\]

This implies

\[
([a]t_0 \langle [a] \square b' / a \rangle \{t'_0 / x\}) \langle \exists_0 / a \rangle \ (\approx^*) c ([a]t_1 \langle [a] \square b' / a \rangle \{t'_1 / x\}) \langle \exists_1 / a \rangle,
\]

i.e., \( E'_0 [t_0.\langle \exists_0 / a \rangle \{t'_0 / x\}] \) \( (\approx^*) c E'_1 [t_1.\langle \exists_1 / a \rangle \{t'_1 / x\}] \) because \( a \) does not occur in \( t'_0, t'_1 \). Renaming \( b \) into \( a \), we obtain

\[
E'_0 [t_0.\langle \exists_0 / a \rangle \{t'_0 / x\}] \ (\approx^*) c E'_1 [t_1.\langle \exists_1 / a \rangle \{t'_1 / x\}]
\]

and because \( u_1 \rightarrow_n^* E'_1 [t_1.\langle \exists_1 / a \rangle \{t'_1 / x\}] \), we have the required result.

\[\square\]

From there, we can prove that \( (\approx^*) c = \approx \) using the usual techniques [7], and then we deduce soundness of \( \approx \). To prove completeness, we show that \( \approx_c \) is an applicative bisimulation.

**Lemma A.4** The relation \( \approx_c \) is a big step applicative bisimulation.
Proof. Suppose $u_0 \simeq_c u_1$. If $u_0 \Downarrow [a] \lambda x. t_0$, then $u_1 \Downarrow [a] \lambda x. t_1$. We have $[a](\mu a. u_0) \rightarrow^* [a] t_0(\langle [a] t/a \rangle \{t/x\})$, $[a](\mu a. u_1) \rightarrow^* [a] t_1(\langle [a] t/a \rangle \{t/x\})$, but also $[a](\mu a. u_0) \simeq_c [a](\mu a. u_1)$. From $\rightarrow^*_n \subseteq \simeq \subseteq \simeq_c$, we have $[a] t_0(\langle [a] t/a \rangle \{t/x\}) \simeq_c [a] t_1(\langle [a] t/a \rangle \{t/x\})$ as wished. \hfill \Box

A.2 David and Py's Counter-Example

Lemma A.5 Let $0 \defeq \lambda x. \lambda y. y$, $1 \defeq \lambda x. \lambda y. x$, and $t_a \defeq \mu c.a.0$. Then we have $\lambda x. \mu a.a.x \mu b.a.x t_a 0 \simeq \lambda x. \mu a.a.x \mu b.a.x t_a 1$.

Proof. We fix a name c, and for all t, we want to relate $[c] \mu a.a.t \mu b.a.t t_a 0$ and $[c] \mu a.a.t \mu b.a.t t_a 1$, which reduce respectively to $[c] t \mu b.c. t_c 0$ and $[c] t \mu b.c. t_c 1$.

Let $s_0^c \defeq \mu b.[c] t_c 0$, $s_1^c \defeq \mu b.[c] t_c 1$, $E_0^c \defeq [c] \Box s_0^c$, and $E_1^c \defeq [c] \Box s_1^c$. We define a relation $[d] t \leadsto^k u$ as $[d] t \rightarrow^*_n u$ if $k = 0$ and as $[d] t \Downarrow [n] [d] \lambda x_1.t_1, [c] t_1 \Downarrow [n] [d] \lambda x_2.t_2, \ldots [c] t_k \rightarrow^*_n u$ for some $t_1, \ldots, t_k$ if $k > 0$. The rationale behind this relation appears in the proof. Note that if $[d] t \leadsto^k u$, then $\nu v(u) \subseteq \{x_1, \ldots, x_k\}$, and $[d] t \langle E/c \rangle \leadsto^k u \langle E/c \rangle$ for all $E$ and $e \neq d$. We define $R$ as

\[
\{(u \langle E/c \rangle \{s_0^c/x_1\}_{i=1}^k, u \langle E/c \rangle \{s_1^c/x_1\}_{i=1}^k) \mid u \in U, t \in T^0, d \notin \nu n(t), [d] t \leadsto^k u\}
\]

\[
\cup \{(c) t_c 0, [c] t_c 1) \mid t \in T^0, d \notin \nu n(t), [d] t \leadsto^k E[x_i], k \geq 1, 1 \leq i \leq k\}
\]

\[
\cup \{(u, u) \mid u \in U^0\}
\]

where $t \{t_i/x_1\}_{i=1}^k$ stands for $t\{t_1/x_1\} \ldots \{t_k/x_k\}$, and we show that $R$ is an applicative bisimulation.

Let $u \langle E/c \rangle \{s_0^c/x_1\}_{i=1}^k R u \langle E/c \rangle \{s_1^c/x_1\}_{i=1}^k$. If $u \rightarrow_n u'$, then it is easy to conclude. Otherwise, we distinguish several cases.

If $u = [c] \lambda y. t_0$ with $e \neq d$, then $[c] t_0([c] \square t/c) \{t'/c\} \langle E''/c \rangle \{s''^c/x_1\}_{i=1}^k R [c] t_0([c] \square t/c) \{t'/c\} \langle E''/c \rangle \{s''^c/x_1\}_{i=1}^k$ (where $t'' = t \langle E''/c \rangle$) for all $t'$, because the relation $[d] t \leadsto^k [e] \lambda y. t_0$ implies $[d] t'' \leadsto^k [e] t_0([c] \square t/c) \{t'/c\}$.

If $u = [d] x_1.t_{k+1}$, then we have the reduction $u \langle E/c \rangle \{s_0^c/x_1\}_{i=1}^k \rightarrow_n [c] t_{k+1} \{s_0^c/x_{k+1}\} \langle E''/c \rangle \{s''^c/x_1\}_{i=1}^k$ as well as the reduction $u \langle E/c \rangle \{s_1^c/x_1\}_{i=1}^k \rightarrow_n [c] t_{k+1} \{s_1^c/x_{k+1}\} \langle E''/c \rangle \{s''^c/x_1\}_{i=1}^k$. We obtain terms in $R$ because we can rewrite them into, respectively, $[c] t_{k+1} \langle E''/c \rangle \{s''^c/x_1\}_{i=1}^k$ and $[c] t_{k+1} \langle E''/c \rangle \{s''^c/x_1\}_{i=1}^k$, and $[d] t \leadsto^k [c] t_{k+1} holds.

Finally, if $u = E[x_i]$, for $1 \leq i \leq k$ (assuming $k \geq 1$), then we have $u \langle E/c \rangle \{s_0^c/x_1\}_{i=1}^k \rightarrow_n [c] t_{t_0}$ and $u \langle E/c \rangle \{s_0^c/x_1\}_{i=1}^k \rightarrow_n [c] t_{t_0}$, and $[c] t_{t_0} R \{c] t_{t_0} R \{c] t_{t_0}$ holds.

Let $[c] t_c 0 R [c] t_c 1$ with $[d] t \leadsto^k E[x_i]$. There exist $t_1, \ldots, t_k$ such that $[d] t \Downarrow [n] [d] \lambda x_1.t_1, [c] t_1 \Downarrow [n] [d] \lambda x_2.t_2, \ldots [c] t_k \Downarrow [n] [d] \lambda x_k.t_k$, and $[c] t_c \rightarrow^*_n E[x_i]$. Then $[c] t_c 0 \rightarrow_n [c] \lambda x_1.t_1(\langle [c] \square t_0/c \rangle \{t_0/x_1\}) t_c 0 \rightarrow_n [c] t_1(\langle [c] \square t/c \rangle \{t_0/x_1\}) 0 \rightarrow_n [c] t_2(\langle [c] \square t/c \rangle \{t_0/x_1\}) \{t_0/x_2\}_{i=1}^k 0 \rightarrow_n [c] t_k(\langle [c] \square t/c \rangle \{t_0/x_1\}) \{t_0/x_2\}_{i=1}^k 0 \rightarrow_n E'[t_c] \rightarrow_n [c] 0$ with $E' \defeq E(\langle [c] \square t_0/c \rangle \{t_0/x_1\}) t_c/x_1 \}^k_{i=1}$. Similarly $[c] t_c 1 \rightarrow_n E''[t_c] \rightarrow_n [c] 0$ with $E'' \defeq E(\langle [c] \square t_1/c \rangle \{t_1/x_1\}) t_c/x_1 \}^k_{i=1}$. Because the two terms evaluate to $[c] 0$, it is easy to conclude. \hfill \Box

64
B Call-by-Value $\lambda\mu$-calculus

B.1 Equivalence Proofs

**Lemma B.1 (see Example 3.6)** The relation $R \overset{\text{def}}{=} \{(u\{v/y\}, u\{\lambda x.\mu b.\{a\}t_0/y\}) \mid [a]t_0 \rightarrow^*_v u\{v/y\}\} \cup \approx$ is an applicative bisimulation.

**Proof.** We proceed by case analysis on $u$. If $u \rightarrow_v u'$ or if $u = [c]a$ for some $c \notin \text{fn}(t_1)$, then we proceed by case analysis on $u$. If $u = [c]a$, then we have to compare (the result of the reduction of) $u(\mathcal{E}/c)\{v(\mathcal{E}/c)/y\}$ and $u(\mathcal{E}/c)\{\lambda x.\mu b.\{a\}t_0(\mathcal{E}/c)/y\}$ for some $\mathcal{E}$ (which depends on the clause we check). The resulting terms are in $R$, because $[a]t_0 \rightarrow^*_v u\{v/y\}$ implies $([a]t_0)(\mathcal{E}/c) \rightarrow^*_v u(\mathcal{E}/c)\{v(\mathcal{E}/c)/y\}$.

We decompose the equivalence proof of Example 3.7 into several lemmas to improve readability. We remind that $u_0 \overset{\text{def}}{=} [b]\lambda x.y.\Omega$, $v \overset{\text{def}}{=} \lambda y.\mu a.\{b\}\lambda x.y$, $u_1 \overset{\text{def}}{=} [b]\lambda x.\Theta_v v y$, and $\Theta_v \overset{\text{def}}{=} \lambda (\lambda x.\Theta_v v y) = (\lambda x.\Theta_v v y)\{\lambda x.y\}$. $\Theta_v(\lambda x.\Theta_v v y)\{\lambda x.y\}$.

**Lemma B.2** Let $t_1, t_2$ be such that $[c]t_1 \not\Downarrow [c]v_1$ and $[d]t_2 \not\Downarrow [d]v_2$ for all $v'$. For all $v'$, we have $[b]\Theta_v v\{[b]\Theta_v v(\downarrow t_1, t_2/h)\} v'$.  

**Proof.** We have 

Similarly, we deduce that $[b]\Theta_v v\{[b]\Theta_v v(\downarrow t_1, t_2/h)\} v'$.

for some $v'_1, v'_2$ (which depend on $v_1, v_2$). So for all $v'$, there exists $v''$ such that $[b]\Theta_v v\{[b]\Theta_v v(\downarrow t_1, t_2/h)\} v' \rightarrow^*_v [b]\Theta_v v\{[b]\Theta_v v(\downarrow t_1, t_2/h)\} v''$; from that, we deduce that $[b]\Theta_v v\{[b]\Theta_v v(\downarrow t_1, t_2/h)\} v'$ is diverging, and therefore $[b]\Theta_v v\{[b]\Theta_v v(\downarrow t_1, t_2/h)\} v'$.

---

**Lemma B.3** Let $v' \overset{\text{def}}{=} \lambda z.t'$, and $t_1$ such that there exists $v_1$ such that $[c]t_1 \not\Downarrow [c]v_1$ for $c \notin \text{fn}(t_1)$. We have 

$$[b]t'\{\lambda y.\Omega/\Omega\} \approx [b]t'\{\lambda y.\Theta_v v\{[b]v'(\downarrow t_1/h)\} y/z\}.$$ 

**Proof.** Let $R \overset{\text{def}}{=} \{(u\{\lambda y.\Omega/\Omega\}, u\{\lambda y.\Theta_v v\{[c]v'(\downarrow t_1)/h\} y/z\}) \mid \exists v_1. [c]t_1 \not\Downarrow [c]v_1, c \notin \text{fn}(t_1), v' = \lambda z.t', \mathcal{E}[t'] \rightarrow^*_v u\} \cup \approx$. We prove $R$ is an applicative bisimilarity, by case analysis on $u$.  

---

65
The case $u \rightarrow_v u'$ is easy. Suppose $u = [d]v_2$, with $v_2 \overset{\text{def}}{=} \lambda z_2.t_2$. Then we have two items to prove.

- Let $[e]t_3 \downarrow_v [e]v_3$ ($e \notin \text{fn}(t_3)$). Then we have to relate

$$[d]t_2\{\lambda y.\Omega/z\}\{[d][t_3/d]/z_2\}$$

and

$$[d]\{[t_2\{[d][t_3/d]/z_2\}]\{\lambda y.\Theta_v u (\langle [v'](\square t_1)\rangle/b) y/z\}\{[d][t_3/d]/z_2\}\{\lambda y.\Theta_v v (\langle [v''](\square t'_1)\rangle/b) y/z\}\}$$

These terms can be rewritten into

$$[d]t_2\{[d][t_3/d]/z_2\}\{\lambda y.\Omega/z\}$$ (B.1)

and

$$[d]t_2\{[d][t_3/d]/z_2\}\{\lambda y.\Theta_v v (\langle [v''](\square t'_1)\rangle/b) y/z\}$$ (B.2)

where $E' \overset{\text{def}}{=} E(\langle [d][t_3/d]\rangle)$, $v'' \overset{\text{def}}{=} v' (\langle [d][t_3/d]\rangle)$, and $t'_1 \overset{\text{def}}{=} t_1 (\langle [d][t_3/d]\rangle)$. We have $v'' = \lambda z.t' (\langle [d][t_3/d]\rangle)$, and

$$E'[t'(\langle [d][t_3/d]\rangle)]$$

$\rightarrow_v [d]v_2 (\langle [d][t_3/d]\rangle) t_3$ (because $E[t'] \rightarrow_v u = [d]v_2$)

$\rightarrow_v [d]v_2 (\langle [d][t_3/d]\rangle) v_3 (\langle [d][v_2 (\langle [d][t_3/d]\rangle) \square/e]\rangle)$ (because $[e]t_3 \downarrow_v [e]v_3$)

$\rightarrow_v [d]t_2 (\langle [d][t_3/d]\rangle) v_3 (\langle [d][v_2 (\langle [d][t_3/d]\rangle) \square/e]\rangle)/z_2$.

The side-conditions are satisfied, therefore (B.1) and (B.2) are in $\mathcal{R}$.

- Let $v_3 = \lambda z_3.t_3$. We have to relate

$$[d]t_3\{v_2 (\langle [d][v_3 \square/d]\rangle)/z_3\}$$

and

$$[d]t_3\{v_2 (\langle [d][v_3 \square/d]\rangle)/z_3\}\{\lambda y.\Omega/z\}$$ (B.3)

These terms can be rewritten into

$$[d]t_3\{v_2 (\langle [d][v_3 \square/d]\rangle)/z_3\}$$

and

$$[d]t_3\{v_2 (\langle [d][v_3 \square/d]\rangle)/z_3\}\{\lambda y.\Theta_v v (\langle [v''](\square t'_1)\rangle/b) y/z\}$$ (B.4)

where $E' \overset{\text{def}}{=} E(\langle [d][v_3 \square/d]\rangle)$, $v'' \overset{\text{def}}{=} v' (\langle [d][v_3 \square/d]\rangle)$, and $t'_1 \overset{\text{def}}{=} t_1 (\langle [d][v_3 \square/d]\rangle)$. We have $v'' = \lambda z.t' (\langle [d][v_3 \square/d]\rangle)$, and

$$E'[t'(\langle [d][v_3 \square/d]\rangle)]$$

$\rightarrow_v [d]v_3 v_2 (\langle [d][v_3 \square/d]\rangle)$ (because $E[t'] \rightarrow_v u = [d]v_2$)

$\rightarrow_v [d]t_3\{v_2 (\langle [d][v_3 \square/d]\rangle)/z_3\}$
The side-conditions are satisfied, therefore (B.3) and (B.4) are in \( \mathcal{R} \).

The last case is when \( u = E'[x v'] \). Then \( u\{\lambda y.\Omega/z\} \rightarrow_v E''[\Omega] \) for some \( E'' \), and

\[
\begin{align*}
  u\{\lambda y.\Theta_v v(\{v'(\square t_1)/b\} y/z) \\
  \rightarrow_v E'(3)[\Theta_v v(\{v'(\square t_1)/b\} v'(3))] \text{ for some } E'(3), v'(3) \\
  \rightarrow_v^* E'[v'(\{v'(\square t_1)/b\} \Theta_v v(\{v'(\square t_1)/b\} y/v'(3))] \\
  \rightarrow_v^* E'[v'(\{v'(\square t_1)/b\} y/z)] \\
  \rightarrow_v^* u\{\lambda y.\Theta_v v(\{v'(\square t_1)/b\} y/z) \text{ because } E[t'] \rightarrow_v^* u
\end{align*}
\]

We obtain two non-terminating terms that are therefore bisimilar. \( \square \)

Lemma B.4 Let \( v' \stackrel{\text{def}}{=} \lambda z.t' \). We have

\[
[b]t'\{\lambda x y.\Omega/z\} \approx [b]t'\{\lambda x y.\Theta_v v(\{b\} v'(\square)/b\} y/z
\]

Proof. Let \( \mathcal{R} \stackrel{\text{def}}{=} \{(u\{\lambda x y.\Omega/z\}, u\{\lambda x y.\Theta_v v(\{v'(\square)/b\} y/z)\} \mid v' = \lambda z.t', E[t'] \rightarrow_v^* u\} \cup \approx 
\). We prove \( \mathcal{R} \) is an applicative bisimilarity, by case analysis on \( u \). The proof is the same as for Lemma B.3; we only detail the last case, where \( u = E'[x v_1 v_2] \). Then \( u\{\lambda x y.\Omega/z\} \rightarrow_v^* E''[\Omega] \) for some \( E'' \), and

\[
\begin{align*}
  u\{\lambda x y.\Theta_v v(\{v'(\square)/b\} y/z) \\
  \rightarrow_v^* E'(3)[\Theta_v v(\{v'(\square)/b\} v_2'(3))] \text{ for some } E'(3), v_2'(3) \\
  \rightarrow_v^* E'[v'(\{v'(\square)/b\} \Theta_v v(\{v'(\square)/b\} y/v_2'(3))] \\
  \rightarrow_v E'[\{\lambda x y.\Theta_v v(\{v'(\square)/b\} y/z)] \\
  \rightarrow_v^* u\{\lambda x y.\Theta_v v(\{v'(\square)/b\} y/z) \text{ because } E[t'] \rightarrow_v^* u
\end{align*}
\]

We obtain two non-terminating terms that are therefore bisimilar. \( \square \)

Lemma B.5 (Example 3.7) The relation \( \{(b)\lambda x y.\Omega, (b)\lambda x y.\Theta_v v y\} \cup \approx \) is an applicative bisimulation.

Proof. We have to prove two items

- Let \( t_1 \) such that \( [c]t_1 \Downarrow_v [c]v_1 \) (\( c \notin \text{fn}(t_1) \)). We have to prove that \( b)\lambda y.\Omega \) is bisimilar to \( b)\lambda y.\Theta_v v(\{b\} \square t_1/b) y \), which, in turn, requires that
  - for all \( d)t_2 \Downarrow_v [d]v_2 \) (\( d \notin \text{fn}(t_2) \), we need \( b)\Omega \approx b)\Theta_v v(\{b\} t_1 t_2/b) v_2' \) for some \( v_2' \). This holds by Lemma B.2;
  - for all \( v' \stackrel{\text{def}}{=} \lambda z.t' \), we need \( b)\{\lambda y.\Omega/z\} \approx b)\{\lambda y.\Theta_v v(\{b\} v'(\square)/b) y/z\} \). This holds by Lemma B.3.

67
• Let \( v' \triangleq \lambda z.t' \). We have to prove that \([b]t'\{\lambda xy.\Omega/z\}\) is related to the term \([b]t'\{\lambda xy.\Theta_v v((b)v' \triangleright \beta) y/z\}\). This a consequence of Lemma B.4.

For the next lemma and its proof, we use the same definitions of terms as in Example 3.3.

**Lemma B.6 (see Example 3.3)** The relation

\[ \mathcal{R} = \{ (u(\{a\} E[v_0(\{a\} E[\square t]/a) \triangleright \beta)/\gamma), u(\{a\} E[v_1 \triangleright \beta]/\gamma) ) \mid [b]t \Downarrow_v [b]v, b \not\in \text{fn}(t) \} \]

is an applicative bisimulation.

**Proof.** We proceed by case analysis on \( u \). If \( u \rightarrow_v u' \), then the result holds. If \( u = [c]v' \) with \( c \neq a \) and \( c \neq b \), then when checking clauses i and ii, we obtain terms that can be written \( u'\{a\} E[v_0(\{a\} E[\square t]/a) \triangleright \beta)/\gamma \} \) and \( u'\{a\} E[v_1 \triangleright \beta]/\gamma \} \) for some \( u' \) and \( \gamma \). Therefore in \( \mathcal{R} \).

If \( u = [a]v' \), then when checking clauses i and ii, we obtain terms that can be written \( u'\{a\} E[v_0(\{a\} E'[\square t]/a) \triangleright \beta)/\gamma \} \) and \( u'\{a\} E'[v_1 \triangleright \beta]/\gamma \} \) (depending on which clause we check). We obtain terms in \( \mathcal{R} \).

If \( u = [b]v' \), then

\[ u(\{a\} E[v_0(\{a\} E[\square t]/a) \triangleright \beta)/\gamma) \rightarrow_v^* [a] E[w' (v'\{a\} E[v_0(\{a\} E[\square t]/a) \triangleright \beta)/\gamma] \lambda x.x)] \]

and \( u(\{a\} E[v_1 \triangleright \beta]/\gamma) \rightarrow_v^* [a] E[u' (v'\{a\} E[v_1 \triangleright \beta] \lambda x.x)] \); we obtain terms in \( \mathcal{R} \).

The next lemma uses the notations of Section 3.3.

**Lemma B.7 (see Example 3.13)** Let \( E, E_0, \) and \( E_1 \) be such that \( E = E_0[E_1] \). Then \( E^\uparrow \approx \lambda x. E_0^\uparrow E_1[x] \).

**Proof.** Let \( a \) be a fresh name. Let \( t \) such that \([d]t \Downarrow_v [d]v\). To check clause i, we have to relate \( [a] u_1. \mathbb{E}[v\{a\} E^\uparrow \triangleright \beta]/d \} \) and \( [a] E_0^\uparrow E_1^\uparrow v\{a\} \lambda x. E_0^\uparrow E_1[x] \triangleright \beta]/d \}. \) These terms reduce respectively to \( \mathbb{E}[v\{a\} E^\uparrow \triangleright \beta]/d \} \) and \( E_0[E_1^\uparrow v\{a\} \lambda x. E_0^\uparrow E_1[x] \triangleright \beta]/d \} \), which we can rewrite into respectively \( u(\{a\} E^\uparrow \triangleright \beta)/d \} \) and \( u(\{a\} \lambda x. E_0^\uparrow E_1[x] \triangleright \beta)/d \}. \) To check clause ii, we have to relate \([a] t \{E^\uparrow/y \} \) and \( [a] E[v\{a\} E^\uparrow \triangleright \beta]/d \} \) for all \( t \). The most interesting case is when \( t = E[y v] \); then these terms reduce to \( \mathbb{E}[v \{E^\uparrow/y \} \} \) and \( \mathbb{E}[v \{\lambda x. E_0^\uparrow E_1[x] \triangleright \beta]/y \} \) respectively. In fact, one can prove the result by showing that

\[ \{ (u(\{a\} E^\uparrow \triangleright \beta)/d), u(\{a\} \lambda x. E_0^\uparrow E_1[x] \triangleright \beta)/d \}, (u(\{E^\uparrow/y \}, u(\{\lambda x. E_0^\uparrow E_1[x] \triangleright \beta]/y \}) \mid E = E_0[E_1] \}) \]

is an applicative bisimulation.
B.2 Proof of Soundness

Lemma B.8 If \([a]t_0 \approx [a]v_1\), then for all \([b]t \rightarrow^* u\) \((b \notin \text{fn}(t))\), we have \(u([a]v_0\langle[a]t \triangledown \rangle \triangledown/b) \approx u([a]v_1\langle[a]t \triangledown \rangle \triangledown/b)\).

Proof. By induction on \(t\) because \(\approx\) has three cases.

Lemma B.9 If \(\lambda x.t_0 \approx^* t_1\), then there exists \(t\) such that \(t_0 \approx^* t\), \(fv(t) \subseteq \{x\}\) and \(\lambda x.t \approx t_1\).

Proof. By induction on \(\lambda x.t_0 \approx^* t_1\).

Lemma B.10 If \([a]t_0 \approx^* u_1\), then there exists a closed \(t\) such that \(t_0 \approx^* t\) and \([a]t \approx u_1\).

Proof. By induction on \([a]t_0 \approx^* u_1\).

Lemma B.11 Let \(t_0 \approx^* t_1\), and \(a \notin \text{fn}(t_0) \cup \text{fn}(t_1)\).

- If \([a]t_0 \rightarrow_n u_0\), then \([a]t_1 \rightarrow_n u_1\) and \(u_0 \approx^* u_1\).
- If \(t_0 = \lambda x.t_0\), then \([a]t_1 \rightarrow_n [a]t_0\lambda x.t_1\) and for all \(t, b, v\) such that \([b]t \rightarrow^* b\) \(v\) and \(b \notin \text{fn}(t)\), we have

\[
[a]t_0'\langle[a]t \triangledown \rangle\{v'[a]x.t_0\langle[a]t \triangledown \rangle \triangledown/b\}/x'\} \approx^* [a]t_1'\langle[a]t \triangledown \rangle\{v'[a]x.t_1\langle[a]t \triangledown \rangle \triangledown/b\}/x'\}
\]

and for all \(v' = \lambda x.t'\), we have

\[
[a]t'\{\lambda x.t_0'\langle[a]v \triangledown \rangle \triangledown/b\}/x\} \approx^* [a]t'\{\lambda x.t_1'\langle[a]v \triangledown \rangle \triangledown/b\}/x\}.
\]

Proof. By induction on \(t_0 \approx^* t_1\). If \(t_0 \approx t_1\), then the result holds by bisimilarity.

If \(t_0 \approx^* t \approx^\circ t_1\), so that \(t\) is closed (using Lemma A.1 if necessary), then we can conclude with the induction hypothesis and the bisimilarity definition.

If \(t_0 \approx^\circ t \approx^\circ t_1\), then we have several cases to consider.

If \(t_0 = \lambda x.t_0\) and \(t_1 = \lambda x.t_1\) with \(t_0' \approx^* t_1'\), we can prove the required result because \(\approx^*\) is substitutive. Suppose \(t_0 = \mu b.u_0\) and \(t_1 = \mu b.u_1\) with \(u_0 \approx^* u_1\).

We have \([a]t_0 \rightarrow v u_0\langle[a]\triangledown/b\}, [a]t_1 \rightarrow v u_1\langle[a]\triangledown/b\}, and \(u_0\langle[a]\triangledown/b\}) \approx^* u_1\langle[a]\triangledown/b\)}\), hence the result holds.

Suppose \(t_0 = t_0^1 t_0^2, t_1 = t_1^1 t_1^2\) with \(t_0^1 \approx^* t_1^1\) and \(t_0^2 \approx^* t_1^2\). We distinguish three cases.

- If \([b]t_0^1 \rightarrow v u_0\) (for some fresh \(b\)), then \([a]t_0 \rightarrow v u_0\langle[a]\triangledown t_0^2/b\). By the induction hypothesis, there exists \(u_1\) such that \([b]t_1^1 \rightarrow v u_1\) and \(u_0 \approx^* u_1\). Consequently, we have \([a]t_1 \rightarrow v u_1\langle[a] t_1^2/b\), and by definition of \(\approx^*\), we have \(u_0\langle[a] t_0^2/b\ \approx^* u_1\langle[a] t_1^2/b\}, as required.

- Suppose \(t_1^1 = \lambda x.t_0^1\) and \([b]t_1^2 \rightarrow v u_0\) for some fresh \(b\); then we have \([a]t_0 \rightarrow v u_0\langle[a]x.t_0^1 \triangledown/b\). By the induction hypothesis, there exists \(u_1\) such that
[b]t_1^2 \rightarrow^*_v u_1 and u_0 \approx^c u_1. Because \lambda x.t'_0 (\approx^*)^c t_1^4, there exists a t such that 
\begin{align*}
t'_0 & \approx^* t \land \lambda x.t \approx^* t_1 
\end{align*}
by Lemma B.9. From \( t'_0 \approx^* t \), we deduce \([a]\lambda x.t'_0 \overline{\square} t_2^2/a \)., which implies \( u_0 ([a] \lambda x.t_0 \overline{\square} t_2^2/a \ (\approx^*)^c u_1 ([a] \lambda x.t_1 \overline{\square} t_2^2/a \overline{\square} b).\)
Because \( \lambda x.t \approx^* t_1 \), there exists \( v_1 \) such that 
\begin{align*}
[a]t_1 & \rightarrow^*_v [a]v_1, \text{ and we have}
[a]l \lambda x.t \approx [a]v_1. \text{ By Lemma B.8, } u_1 ([a] \lambda x.t (a) \overline{\square} t_2^2/a) \approx u_1 ([a]v_1 (a) \overline{\square} t_2^2/a \overline{\square} b) \text{ holds, so}
[u_0 ([a] \lambda x.t_0 \overline{\square} b) \approx^c u_1 ([a]v_1 (a) \overline{\square} t_2^2/a \overline{\square} b) \text{ holds as well. Besides, we have}
[a]t_1 \rightarrow^*_v u_1 ([a]v_1 (a) \overline{\square} t_2^2/a \overline{\square} b), \text{ hence we have the required result.}
\end{align*}

- If \( t_1^4 = \lambda x.t_0^6 \) and \( t_0^6 = \lambda y.t_0^6 \), then \([a]t_0 \rightarrow^*_v [a]t_0^6 \overline{\square} x/y \). By Lemma B.9, there exist closed \( t, t' \) such that \( t_0 \approx^* t, \lambda x.t \approx^* t_1 \), \( t_0' \approx^* t' \), and \( \lambda y.t \approx^* t_2 \). From \( t_0' \approx^* t' \), we deduce \( t_0' \approx^* t \overline{\square} t_1^4/a \), and from \( t_0' \approx^* t' \), we deduce \( \lambda y.t_0' \approx^* \lambda y.t_1 \overline{\square} t_2^2/a \overline{\square} a \). Consequently, we have \([a]t_0' ([a] \lambda y.t_0' \overline{\square} x/y) \approx^* [a]t_0 ([a] \lambda y.t_1 \overline{\square} t_2^2/a) \). From \( \lambda y.t \approx^* t_2 \), by bisimilarity, there exists \( v_1 \) such that \([a]t_2 \rightarrow^*_v [a]v_1 \), and 
\begin{align*}
[a]t ([a] \overline{\square} t_1^4/a \overline{\square} x/y) \approx [a]t ([a] \overline{\square} t_1^4/a) \{v_1 ([a] \lambda x.t ([a] \overline{\square} t_2^2/a) \overline{\square} a) / x \}
\end{align*}
holds (clause ii). From \( \lambda x.t \approx^* t_1 \), by bisimilarity, there exists \( t_1 \) such that \([a]t_1^4 \rightarrow^*_v [a]l \lambda x.t_1, \text{ and}
\begin{align*}
[a]t ([a] \overline{\square} t_1^4/a) \{v_1 ([a] \lambda x.t ([a] \overline{\square} t_2^2/a) \overline{\square} a) / x \} \approx [a]t_1 ([a] \overline{\square} t_1^4/a) \{v_1 ([a] \lambda x.t_1 ([a] \overline{\square} t_2^2/a) \overline{\square} a) / x \}
\end{align*}
holds (clause i). Finally, we have 
\begin{align*}
[a]t_0' ([a] \lambda y.t_0' \overline{\square} x/y) \approx^* [a]t_1 ([a] \overline{\square} t_1^4/a) \{v_1 ([a] \lambda x.t_1 ([a] \overline{\square} t_2^2/a) \overline{\square} a) / x \},
\end{align*}
and because \([a]t_1 \rightarrow^*_v [a]t_1 ([a] \overline{\square} t_1^4/a) \{v_1 ([a] \lambda x.t_1 ([a] \overline{\square} t_2^2/a) \overline{\square} a) / x \}, \text{ we have the required result.}

Suppose \( t_0 = t_0' (\overline{\square} b) \), \( t_1 = t_1' (\overline{\square} b) \) with \( t_0' \approx^* t_1' \), \( \overline{\square} b \approx^* \overline{\square} b \). If \([a]t_0 \rightarrow^*_v u_0 \), then in fact \([a]t_0' \rightarrow^*_v u_0' \) with \( u_0' = u_0' (\overline{\square} b) \). By the induction hypothesis, there exists \( u_1' \) such that \([a]t_1' \rightarrow^*_v u_1' \) and \( u_0' \approx^* u_1' \). Consequently, we have \([a]t_1 \rightarrow^*_v u_1' (\overline{\square} b) \), and \( u_0' (\overline{\square} b) \approx^* u_1' (\overline{\square} b) \) holds, as wished. If \([a]t_0 \) is a named value, then in fact \( t_0' = \lambda x.t_0'' \) and \( t_0 = \lambda x.t_0'' \overline{\square} b \). The result holds by using the induction hypothesis (and with some renaming of \( b \) into a fresh \( c \) to avoid name clashes).

\[ \square \]

**Lemma B.12** The relation \( \approx^c \) is an applicative simulation.

**Proof.** Let \( u_0 \approx^c u_1 \); we prove the simulation clause by induction on \( u_0 \approx^c u_1 \).

If \( u_0 \approx u_1 \), then the result holds by bisimilarity. If \( u_0 \approx^* u \approx^c u_1 \), then we can conclude using the induction hypothesis and the bisimilarity definition.

If \( u_0 \approx^c u_1 \), then we have two cases. Suppose \( u_0 = [a]t_0 \) and \( u_1 = [a]t_1 \) with \( t_0 \approx^c t_1 \). If \( a \notin \text{fn}(t_0) \cup \text{fn}(t_1) \), then we can apply Lemma B.11 directly to get the

70
required result. Otherwise, let \( b \notin \text{fn}(t_0) \cup \text{fn}(t_1) \). If \( u_0 \rightarrow_v u_0' \), then \( [b]t_0 \rightarrow_v u_0' \) and \( u_0' = u_0' ([a] \square / b) \). We can apply Lemma B.11 to \([b]t_0\) and \([b]t_1\), and then rename \( b \) into \( a \).

Suppose \( u_0 = u_0' (\mathbb{E}_0 / a) \), \( u_1 = u_1' (\mathbb{E}_1 / a) \) with \( u_0' (\approx^*)^c u_1' \) and \( \mathbb{E}_0 (\approx^*)^c \mathbb{E}_1 \).

If \( u_0 \) is a named value, then we use the induction hypothesis, with some renaming (as in the call-by-name case) to avoid some name clashes.

If \( u_0 \rightarrow_v \), then again we distinguish several cases. First, suppose \( u_0' \rightarrow_v u_0'' \); then \( u_0 \rightarrow_v u_0'' (\mathbb{E}_0 / a) \). By the induction hypothesis, there exists \( u'' \) such that \( u_1' \rightarrow_v^* u'' \) and \( u_0'' (\approx^*)^c u_0'' \). Then \( u_1 \rightarrow_v^* u_1' (\mathbb{E}_1 / a) \) and \( u_0'' (\mathbb{E}_0 / a) (\approx^*)^c u_0'' (\mathbb{E}_1 / a) \), hence the result holds.

Second, assume \( u_0'' = [a] \lambda x. t_0 \), \( \mathbb{E}_0 = \mathbb{E}_0' [\square ] v_0 \) and \( u_0 \rightarrow_v \mathbb{E}_0' [t_0 (\mathbb{E}_0 / a) \{v_0 / x\}] \). By Lemma B.10, there exists \( t \) such that \( \lambda x. t_0 \approx^* \lambda x. t \) and \( [a] t \approx u_1' \). By Lemma B.9, there exists \( t' \) such that \( t_0 \approx^* t' \) and \( \lambda x. t' \approx t \). We can prove by induction on \( \mathbb{E}_0 (\approx^*)^c \mathbb{E}_1 \) that \( E_1 = E_1' [\square ] t_1' \) with \( E_0' (\approx^*)^c E_1' \) and \( v_0 (\approx^*)^c t'_1 \). Suppose \( v_0 = \lambda x. t_0' \). By Lemma B.9, there exists \( s \) such that \( t_0' \approx^* s \) and \( \lambda x. s \approx t_1' \). Let \( E_1', s', t_1' \) be \( E_1', s \), and \( t_1' \) with \( a \) renamed into a fresh \( c \). From \( t_0 \approx^* t' \), \( \lambda x. t_0' \approx^* t_1' \), and \( \lambda x. t_0'' \approx^* \lambda x. s' \), we deduce

\[
[a] t_0 \langle [a] \square \lambda x. t_0' / a \rangle \{ \lambda x. t_0'' / x \} (\approx^*)^c [a] t' \langle [a] \square t_1' / a \rangle \{ \lambda x. s' / x \}.
\]

Because \( \lambda x. s' \approx t_1'' \), there exists \( v_1' \) such that \( [b] t_1'' \rightarrow_v^* [b] v_1' \) (for a fresh \( b \)), and

\[
[a] t' \langle [a] \square t_1' / a \rangle \{ \lambda x. s' / x \} \approx [a] t' \langle [a] \square t_1' / a \rangle \{ v_1' \langle [a] \lambda x. t_1' \langle [a] \square t_1' / a \rangle / b \} / x \}
\]

(using the second item of the value case of the bisimulation definition). Because \( \lambda x. t' \approx t \), there exists \( t'' \) such that \( [a] t \rightarrow_v^* [a] \lambda x. t'' \) and

\[
[a] t' \langle [a] \square t_1' / a \rangle \{ v_1' \langle [a] \lambda x. t' \langle [a] \square t_1' / a \rangle / b \} / x \}
\]

(by clause i). Because \( [a] \lambda x. t' \approx u_1' \), there exists \( t_1 \) such that \( u_1' \rightarrow_v^* [a] \lambda x. t_1 \) and we have also

\[
[a] t'' \langle [a] \square t_1' / a \rangle \{ v_1' \langle [a] \lambda x. t'' \langle [a] \square t_1' / a \rangle / b \} / x \}
\]

(by clause i). Finally, we obtain

\[
[a] t_0 \langle [a] \square \lambda x. t_0' / a \rangle \{ \lambda x. t_0'' / x \} (\approx^*)^c [a] t_1 \langle [a] \square t_1' / a \rangle \{ v_1' \langle [a] \lambda x. t_1 \langle [a] \square t_1' / a \rangle / b \} / x \}
\]

by transitivity of \( \approx \) and definition of \( \approx^* \), from which we can deduce

\[
E_0' [t_0 \langle E_0' [\square ] \lambda x. t_0' / a \rangle \{ \lambda x. t_0'' / x \}]
\]

\[
(\approx^*)^c E_1' [t_1 \langle E_1' [\square ] t_1' / a \rangle \{ v_1' \langle E_1' [\lambda x. t_1 \langle [a] \square t_1' / a \rangle / b \} / x \}].
\]
Renaming $c$ back into $a$, we get

$$
\mathbb{E}'_0[t_0\langle \mathbb{E}'_i[\lambda x.t'_0]/a \rangle \{\lambda x.t'_0/x\}]
$$

$$(\approx^* \Rightarrow) \mathbb{E}'_1[t_1\langle \mathbb{E}'_i[\lambda x.t'_1]/a \rangle \{v_1\langle \mathbb{E}'[\lambda x.t(\mathbb{E}'_i[\lambda x.t'_1]/a) \square/b]/x\}]
$$

(assuming $v_1$ is the result of renaming $c$ into $a$ in $v'_1$), which is the same as

$$
\mathbb{E}'_0[t_0\langle \mathbb{E}_a/a \rangle \{\lambda x.t'_0/x\}]
$$

$$(\approx^* \Rightarrow) \mathbb{E}'_1[t_1\langle \mathbb{E}_a/a \rangle \{v_1\langle \mathbb{E}'[\lambda x.t(\mathbb{E}'_i[\lambda x.t'_1]/a) \square/b]/x\}]
$$

One can check that $u_1 \rightarrow^*_v \mathbb{E}'_1[t_1\langle \mathbb{E}_a/a \rangle \{v_1\langle \mathbb{E}'[\lambda x.t(\mathbb{E}'_i[\lambda x.t'_1]/a) \square/b]/x\}]]$, hence the result holds.

Finally, the last case is $u'_0 = [a]v_0$, $\mathbb{E}_0 = \mathbb{E}'_0[\lambda x.t_0 \square]$, which give $u_0 \rightarrow^*_v \mathbb{E}'_0[t_0\langle \mathbb{E}_0/a \rangle/x]$. This case is similar to the previous one and is left to the reader.

$\square$
On Grainless Footprint Semantics for Shared-memory Programs

Stephen Brookes

Carnegie Mellon University

1 Outline

We develop an improved grainless denotational semantics for shared-memory parallel programs, building on ideas from earlier trace-based models with local states and footprints [4]. The key new idea is a more refined approach to race detection, leading to a model with better abstraction properties. Rather than treat a race condition as a “global” catastrophe [3,4], we track information about variables whose value may be tainted by a race, and retain accurate information about unaffected variables. As in the prior work, we abstract away from state changes that occur in between synchronization points, in a manner consistent with Dijkstra’s Principle [5]. Our semantics supports compositional program analysis based on “sequential” reasoning for sequential code fragments, even when this code occurs in parallel contexts, and yields a simple semantic characterization of race-free code. The semantics validates the static constraints on “critical variables” imposed in concurrent programming methodology [6,9] and serves as a foundation for reasoning about safe partial correctness, as in concurrent separation logic [8]. The new treatment of race detection allows for more refined analysis of racy programs. By framing our ideas and concepts in a general manner we hope that our results may be applied in a wider setting.

Introduction

Shared-memory programs are difficult to reason about, because of the potential for concurrent interference. Early semantic models assume fixed granularity of exe-

---

1 This research was sponsored by the National Science Foundation under grant no. CCF-1017011. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.
duction, for example atomic assignments or atomic reads and writes [2,10]. Such
assumptions may not hold in practice, and program analysis based on such models
is only valid for implementations enforcing the required degree of granularity. In
contrast, traditional semantic models for sequential programs ignore granularity and
focus on the relationship between initial and final states of a program execution,
interpreting a program as a state transformation. Such semantics is simple and
easy to use for compositional reasoning about sequential programs, but inadequate
for parallel programs [10]: the state transformation denoted by $c_1 \parallel c_2$ cannot be
determined from the state transformations denoted by $c_1$ and $c_2$. Ideally we need
a semantic model for parallel programs that enjoys the simplicity of state trans-
formation semantics while retaining enough information about intermediate states
to allow proper modeling of concurrent interference, without making granularity
assumptions.

Such concerns have stimulated an effort to design a “grainless” semantic model
Reynolds sought to avoid granularity by breaking atomic actions into instantaneous
fragments, an approach that leads to a semantics based on very small steps, and
therefore likely to suffer from combinatorial problems. This author developed a
“footstep trace” model, a pre-cursor to the approach offered in this paper, but in
retrospect we can now see that this model is overly complex and fails full abstraction.
Here we offer a more streamlined version of footstep trace semantics, with better
abstraction properties. The main new idea involves a more refined account of race
conditions, an apparently simple idea with deep ramifications in the construction
of the semantics. We classify our semantic model as “grainless” because the model
construction abstracts away from irrelevant scheduling details in such a way that
there is no need to retain information about what constitutes an “atomic” action,
other than the usual assumption that primitive operations for synchronization are
atomic.

We deal here with a simple shared-memory language, omitting pointers. Our
development builds on our prior work on trace models [2] and on concurrent sep-
ARation logic [3,4], in which we combined mutable state with concurrency, so we
expect to be able to adapt the new ideas presented here accordingly. It is straight-
forward to incorporate locally scoped declarations. For space reasons we defer these
extensions and some semantic details and proofs to a fuller version of the paper.

2 Syntax and Static Semantics

We will deal with a simple shared-memory parallel language, extending the familiar
class of sequential while-programs with parallel composition and a well known
synchronization construct: conditional critical regions. Identifiers (or program vari-
ables) $i$ are assignable integer-valued variables, and region names (or resources) $r$
take values 0 and 1, representing “available” and “unavailable”. The sets $\text{Ide}$ of
identifiers and $\text{Res}$ of resource names are disjoint. The syntax of integer expres-
sions $e$ and boolean expressions $b$ is conventional, including the usual arithmetic and
boolean constructs. We define the sets \( \text{free}(e) \) and \( \text{free}(b) \) of identifiers having a free syntactic occurrence in \( e \) and \( b \), as usual, by structural induction.

The syntax of commands (or processes) \( c \) is given by:

\[
c ::= \text{skip} \mid i := e \mid c_1 ; c_2 \mid \text{if } b \text{ then } c_1 \text{ else } c_2 \mid \text{while } b \text{ do } c \mid c_1 \parallel c_2 \mid \text{with } r \text{ when } b \text{ do } c
\]

Resources behave like binary semaphores with atomic operations for locking and unlocking, so critical regions can be used to ensure mutually exclusive access to shared variables. As usual we abbreviate \( \text{with } r \text{ when true do } c \) as \( \text{with } r \text{ do } c \).

Resource names are not assignable. The sets \( \text{res}(c) \) and \( \text{free}(c) \) of resource names and identifiers with a free occurrence in \( c \) are defined by structural induction in the obvious manner.

## 3 Dynamic Semantics

A program denotes a set of traces representing interactive computations in which the program and its environment make changes to the shared state. Each step in a trace represents the effect of a finite sequence of actions performed by the program, and records just the overall footprint. We detect the potential for race conditions, involving a write to a shared variable whose value is used in a concurrent update. A race condition can lead to unpredictable behavior, so we use \( \top \) to represent the value of a variable whose value is race-dependent, a special value which taints all future computations involving that variable. In this sense we treat races as “locally” catastrophic, and we track accurately the values of variables unaffected by races.

### States

States are finite partial functions from identifiers and resource names to values. For simplicity we let \( V \) be the set of integers and use values 0 (“available”) and 1 (“unavailable”) for resources. Let \( V^\top = V \cup \{\top\} \). Let \( \text{Var} = \text{Ide} \cup \text{Res} \). We use \( \iota \) to range over \( \text{Var} \), \( i \) over \( \text{Ide} \) and \( r \) over \( \text{Res} \). We use a list-like notation \([\iota_1 : v_1, \ldots, \iota_k : v_k]\) for states, and we may also use set-theoretic notation such as \( \{\langle \iota_1, v_1\rangle, \ldots, \langle \iota_k, v_k\rangle\}\).

**Definition 1** The set \( \Sigma \) of states is given by:

\[
\Sigma = \{\sigma : \text{Var} \rightarrow_{\text{fin}} V^\top \mid \forall r \in \text{dom}(\sigma) \cap \text{Res}. \sigma(r) \in \{0, 1\}\}.
\]

We use \( \sigma \) and \( \tau \) to range over states, and let \( \text{res}(\sigma) = \text{dom}(\sigma) \cap \text{Res} \) be the set of resources used in \( \sigma \). For a set \( X \subseteq \text{Var} \) we let \( \sigma \setminus X = \{\langle \iota, v\rangle \in \sigma \mid \iota \notin X\} \) and \( \sigma \upharpoonright X = \{\langle \iota, v\rangle \in \sigma \mid \iota \in X\} \). When \( X \) is a singleton we write \( \sigma \setminus \iota \) for \( \sigma \setminus \{\iota\} \).

**Definition 2** A state \( \sigma \) is race-free if \( \top \notin \text{rge}(\sigma) \).

**Definition 3** Two states \( \sigma \) and \( \tau \) are consistent, \( \sigma \upharpoonright \tau \), iff they agree on the values of all relevant variables, i.e. \( \forall \iota \in \text{dom}(\sigma) \cap \text{dom}(\tau). \sigma(\iota) = \tau(\iota) \).
Consistency is the same as requiring that \( \sigma \upharpoonright \text{dom}(\tau) = \tau \upharpoonright \text{dom}(\sigma) \), or that \( \sigma \cup \tau \) is also a well-defined state. States with disjoint domains are always consistent.

**Definition 4** We let \([\sigma \mid \iota \mapsto v] = (\sigma \setminus \iota) \cup \{(\iota, v^\top)\}\), where \(v^\top = \bot\) if \((\iota, \top) \in \sigma\), and \(v^\top = v\) otherwise. This is the state obtained by updating \(\sigma\) with \(\iota : v\), unless \(\sigma(\iota) = \bot\), in which case the update has no effect.

We generalize to multiple updates, writing \([\sigma \mid \tau]\) for the state obtained by updating \(\sigma\) with the updates in \(\tau\), and \([\sigma \mid X \mapsto \top]\) for updating \(\sigma\) with \(\{(x, \top) \mid x \in X\}\).

Updating is associative, i.e. \([\sigma \mid \tau \mid \rho] = [\sigma \mid [\tau \mid \rho]]\), so we can write \([\sigma \mid \tau \mid \rho]\) without ambiguity.

**Steps**

Steps represent the effect of state changes, and to model footprints of programs we record just the portion of state relevant to a step, rather than the entire global state. So a step will involve a pair of states \((\sigma, \sigma')\), where \(\sigma\) is the piece of state read and \(\sigma'\) is the piece of state written, and we decorate this pair with a flow relation \(R \subseteq \text{dom}(\sigma) \times \text{dom}(\sigma')\) indicating in particular which reads influence the value of each write. We require that if \((\iota, \top) \in \sigma \& (\iota, \iota') \in R\) then \((\iota', \top) \in \sigma'\), since an update based on a tainted value is also deemed to be tainted. Further, if \(\iota \in \text{dom}(\sigma')\) we insist that \((\iota, \iota) \in R\), since a write can only be performed if its target is present in the initial state. Coupled with the previous requirement this means that once a variable has been involved in a race its value never “recovers”.

Note that \(\text{dom}(\sigma') \subseteq \text{dom}(R^{-1})\): for every variable \(\iota\) written in the step the set \(R^{-1}(\iota) \subseteq \text{dom}(\sigma)\) indicates the variables whose values in \(\sigma\) influence the update.

**Definition 5** The set \(\Lambda\) of steps consists of all triples \((\sigma, R, \sigma')\) with \(\sigma, \sigma' \in \Sigma\), such that:

- \(R \subseteq \text{dom}(\sigma) \times \text{dom}(\sigma')\).
- For all \(\iota \in \text{dom}(\sigma')\), \((\iota, \iota) \in R\).
- If \((\iota, \top) \in \sigma\) and \((\iota, \iota') \in R\) then \((\iota', \top) \in \sigma'\).

We use \(\lambda\) and \(\mu\) to range over the set of steps.

**Definition 6** For a step \(\lambda = (\sigma, R, \sigma')\), let \(\text{reads}(\lambda) = \text{dom}(\sigma)\), \(\text{writes}(\lambda) = \text{dom}(\sigma')\), and \(\text{res}(\lambda) = \text{dom}(\sigma) \cap \text{Res}\). By assumption, \(\text{writes}(\lambda) \subseteq \text{reads}(\lambda)\).

For a step \((\sigma, R, \sigma')\) and \(\iota \in \text{dom}(\sigma')\), \(R^{-1}(\iota)\) is the set of variables used to compute the update for \(\iota\). Since \(R\) is surjective, we can specify the dependency relation by listing \(R^{-1}(\iota)\) for each \(\iota \in \text{dom}(\sigma')\). We omit the flow relation when we intend the smallest relation satisfying the requirements, often the identity relation on \(\text{dom}(\sigma')\) or the empty relation when \(\text{dom}(\sigma') = \{\}\). Where relevant we use \(U\) for the universal relation on \(\text{dom}(\sigma) \times \text{dom}(\sigma')\). It is helpful to introduce names for some simple steps.
Brookes

In each case the intended flow relation is obvious:

\[
\begin{align*}
\text{lock}(r) &= ([r : 0], [r : 1]) \\
\text{unlock}(r) &= ([r : 1], [r : 0]) \\
\text{read}(i, v) &= ([i : v], []) \\
\text{write}(i, v, v') &= ([i : v], [i : v']) \\
\delta &= ([], [])
\end{align*}
\]

$\delta$ is an “idle” step. For steps in which multiple reads and writes occur, there may be several possible choices of flow relation, expressing different dependencies. For example, in the step $([x : 0, y : 0], \{(y, y)\}, [y : 1])$ the update to $y$ does not depend on the read of $x$, whereas in $([x : 0, y : 0], \{(x, y), (y, y)\}, [y : 1])$ the update to $y$ depends on both $x$ and $y$. There are also racy steps such as $([x : 0], [x : ?])$ in which a race condition affects the value of an identifier. Note that $([x : ?, y : 0], [x : ?, y : 1])$ is also a valid step, but that $([x : ?, y : 0], [x : 0, y : 1])$ is not.

In the forthcoming semantic development, steps involving resources play a special rôle consistent with the underlying assumption that operations to lock and unlock resources are atomic actions. Steps with $\text{res}(\lambda) = \text{res}(\mu) = \emptyset$ may be composable either consecutively or concurrently, to produce a single step representing the composite effect, whereas a step with $\text{res}(\lambda) \neq \emptyset$ is atomic.

**Executing steps**

Although steps describe footprints, programs operate on a shared global state, and we need to characterize the effect on the global state of executing a step. Global states are also finite partial functions from variables to values, so $\Sigma$ as defined before also represents the set of global states.

Given a state $\sigma$ we can characterize the steps that are *executable* from $\sigma$, and their *effect*, with an enabling relation $\Rightarrow \subseteq \Sigma \times \Lambda \times \Sigma$. We write this as an infix relation, writing $\sigma \Rightarrow \rightarrow \sigma'$ when step $\lambda$ is enabled from $\sigma$, and its execution causes the state to change to $\sigma'$. The definition is intuitive: a step can only be executed from a state consistent with its reads, and its effect is described by its writes, leaving all other variables unchanged. We say that state $\sigma$ *enables* step $\lambda$ if $\sigma$ satisfies the read properties required to execute $\lambda$, i.e. if $\text{reads}(\lambda) \subseteq \sigma$.

**Definition 7** The enabling relation $\Rightarrow \subseteq \Sigma \times \Lambda \times \Sigma$ is given by:

\[
\sigma \overset{(\tau, R, \tau')}{\Rightarrow} \sigma' \iff \tau \subseteq \sigma & \tau' = [\sigma | \tau'].
\]
Referring again to the examples introduced above, note the key facts that

\[
\begin{align*}
\sigma & \xrightarrow{\text{read}(i,v)} \sigma \quad \text{iff} \ i : v \in \sigma \\
\sigma & \xrightarrow{\text{write}(i,v,v')} [\sigma \mid i : v] \quad \text{iff} \ i : v \in \sigma \\
\sigma & \xrightarrow{\text{lock}(r)} [\sigma \mid r : 1] \quad \text{iff} \ r : 0 \in \sigma \\
\sigma & \xrightarrow{\text{unlock}(r)} [\sigma \mid r : 0] \quad \text{iff} \ r : 1 \in \sigma
\end{align*}
\]

Also note that lock(r) is not enabled in \( \sigma \) if \( \sigma(r) = 1 \), and unlock(r) is not enabled in \( \sigma \) if \( \sigma(r) = 0 \). It is obvious that \( \sigma \xrightarrow{\Delta} \sigma' \) if \( \sigma \xrightarrow{\Delta} \sigma \) always holds. In general when \( \sigma \xrightarrow{\Delta} \sigma' \) we have \( \text{dom}(\sigma') = \text{dom}(\sigma) \).

**Consecutive steps**

Two steps \( \lambda = (\sigma, R, \sigma') \) and \( \mu = (\tau, S, \tau') \) are **consecutive** (or sequentially executable) if their effects are composable, \( \lambda \) first followed by \( \mu \). This is the case when \( \mu \) follows \( \lambda \), i.e. \( \mu \) can be enabled after \( \lambda \), as characterized below.

**Definition 8** For steps \( \lambda = (\sigma, R, \sigma') \) and \( \mu = (\tau, S, \tau') \) we say that \( \mu \) follows \( \lambda \), written \( \lambda \sim \mu \), if \( \tau \uparrow [\sigma \mid \sigma'] \).

The requirement that \( \tau \uparrow [\sigma \mid \sigma'] \) says that the reads of \( \mu \) are consistent with the effect of \( \lambda \). This is equivalent to requiring that \( \sigma \uparrow (\tau \setminus \text{dom}(\sigma')) \) and \( \tau \uparrow \sigma' \). Note that the sequential composition operation on steps is **partial**, only defined on steps that satisfy the imposed constraints.

**Definition 9** When \( \lambda = (\sigma, R, \sigma') \) and \( \mu = (\tau, S, \tau') \) and \( \lambda \sim \mu \), we define

\[
\lambda; \mu = (\sigma \cup (\tau \setminus \text{dom}(\sigma'))), \ R; S, \ [\sigma' \mid \tau' \mid \rho],
\]

where \( \rho = \{(i', \tau) \mid \exists (i, \tau) \in \sigma'. (i, i') \in S\} \), and where \( R; S \) is the relation

\[
\begin{align*}
\{&(i, i') \mid \exists i'. (i, i') \in R \land (i', i'') \in S \} \\
\cup &\ \{ (i, i') \in R \mid i' \not\in \text{dom}(S) \} \\
\cup &\ \{(i', i'') \in S \mid i' \not\in \text{rg}(R) \}
\end{align*}
\]

The term \( \rho \) here propagates the effect of racy updates from the first step. If \( \lambda \) is race-free \( \rho \) degenerates to the empty state, and the write effect of the cumulative step is \( [\sigma' \mid \tau] \) as expected. The flow relation \( R; S \) composes \( R \) and \( S \), bearing in mind that the first step may write to a variable not used in the second step, and the second step may read a variable not influenced by the first step.

**Lemma 10** \( \lambda_1 \sim \lambda_2 \& (\lambda_1; \lambda_2) \sim \lambda_3 \) iff \( \lambda_2 \sim \lambda_3 \& \lambda_1 \sim (\lambda_2; \lambda_3) \). When these equations hold \( \lambda_1; (\lambda_2; \lambda_3) = (\lambda_1; \lambda_2); \lambda_3 \). Further, \( \delta = ([], []) \) is a unit for sequential composition: \( \lambda \sim \delta \sim \lambda \) and \( \lambda; \delta = \delta; \lambda = \lambda \) for all \( \lambda \in \Lambda \).

Proof: An easy calculation using the definitions. For all relations \( R, S, T \) we have \( (R; S); T = R; (S; T) \), and \( R; \{\} = \{\}; R = R \) for all \( R \).
Examples
In each of the following cases, the two steps are sequentially executable and we show the resulting step. As usual the intended flow relations are obvious:

\[
\begin{align*}
\text{read}(x, v) ; \text{read}(x, v) &= \text{read}(x, v) \\
\text{read}(x, v_1) ; \text{read}(y, v_2) &= ([x : v_1, y : v_2], []) \\
\text{read}(x, v) ; \text{write}(x, v, v') &= \text{write}(x, v, v') \\
\text{write}(x, v, v') ; \text{read}(x, v) &= \text{write}(x, v, v') \\
\text{write}(x, v_1, v_2) ; \text{write}(x, v_2, v_3) &= \text{write}(x, v_1, v_3) \\
\text{write}(x, v_1, v'_1) ; \text{write}(y, v_2, v'_2) &= ([x : v_1, y : v_2], [x : v'_1, y : v'_2])
\end{align*}
\]

In particular, reads and writes to distinct identifiers have the same effect in either sequential order. Note also that writes to the same identifier are only sequentially composable if the value read by the second step agrees with the value written in the first step.

As expected, there is a simple relationship between sequential composition of steps and execution.

**Lemma 11** If \( \lambda \sim \mu \), then \( \sigma \xrightarrow{\lambda, \mu} \sigma'' \) iff \( \exists \sigma'. (\sigma \xrightarrow{\lambda} \sigma' \& \sigma' \xrightarrow{\mu} \sigma'') \).

Concurrent steps
Steps \( \lambda \) and \( \mu \) with \( \text{res}(\lambda) \cap \text{res}(\mu) = \{\} \) and \( \text{reads}(\lambda) \uparrow \text{reads}(\mu) \) are said to be **concurrently executable**, or **concurrent** for short; they have compatible reads so there is a state from which both steps are enabled, and since they use different resources they can be executed simultaneously without violating the mutual exclusion constraints on resource use. The steps **conflict** if one writes to an identifier upon which the other one depends. When this happens we record the value of the race-sensitive identifier as \( \top \), using the flow relations as a guide. In this way we obtain a composite step \( \lambda \otimes \mu \) describing the concurrent combination of \( \lambda \) and \( \mu \). Again this operation on steps is **partial**, only defined for steps with compatible read requirements and needing disjoint resources.

**Definition 12** Two steps \( \lambda = (\sigma, R, \sigma') \) and \( \mu = (\tau, S, \tau') \) are concurrently executable, \( \lambda \text{ co } \mu \), if \( \sigma \uparrow \tau \& \text{res}(\sigma) \cap \text{res}(\tau) = \{\} \). When this holds we define

\[
\lambda \otimes \mu = (\sigma \cup \tau, R \cup S, [(\sigma' \cup \tau') | X \mapsto \top]),
\]

where \( X \) is the set of identifiers whose value is susceptible to a race condition, i.e. those identifiers written by one of the steps with a value dependent on an identifier affected by the other step:

\[
X = \{i \in \text{dom}(\sigma') | R^{-1}(i) \cap \text{dom}(\tau') \neq \{\}\}
\cup \{i \in \text{dom}(\tau') | S^{-1}(i) \cap \text{dom}(\sigma') \neq \{\}\}.
\]
Since $\sigma$ and $\tau$ are assumed to be compatible ($\sigma \upharpoonright \tau$) the union $\sigma \cup \tau$ is a partial function, hence a valid state. The flow relation $R \cup S$ combines the flow information from the two steps in the obvious way. Even though the union $(\sigma' \cup \tau')$ may not be a partial function when $\text{dom}(\sigma') \cap \text{dom}(\tau') \neq \emptyset$, the update $X \mapsto \top$ does produce a partial function, hence a valid state, since $X \supseteq \text{dom}(\sigma') \cap \text{dom}(\tau')$, $\text{dom}(\sigma') \subseteq \text{dom}(R^{-1})$, and $\text{dom}(\tau') \subseteq \text{dom}(S^{-1})$.

We can paraphrase the above definition, perhaps in a more readable manner, as saying that $\lambda \otimes \mu = (\sigma \cup \tau, R \cup S, \theta)$, where $\theta$ is the state with $\text{dom}(\theta) = \text{dom}(\sigma') \cup \text{dom}(\tau')$ and the following properties:

- $\theta(i) = \sigma'(i)$ if $i \in \text{dom}(\sigma')$ and $R^{-1}(i) \cap \text{dom}(\tau') = \emptyset$;
- $\theta(i) = \tau'(i)$ if $i \in \text{dom}(\tau')$ and $S^{-1}(i) \cap \text{dom}(\sigma') = \emptyset$;
- $\theta(i) = \top$ otherwise.

**Lemma 13** $\lambda_1 \text{ co } \lambda_2$ iff $\lambda_2 \text{ co } \lambda_1$, and when these hold $\lambda_1 \otimes \lambda_2 = \lambda_2 \otimes \lambda_1$. Further, $(\lambda_1 \text{ co } \lambda_2$ and $(\lambda_1 \otimes \lambda_2) \text{ co } \lambda_3)$ iff $(\lambda_2 \text{ co } \lambda_3$ and $\lambda_1 \text{ co } (\lambda_2 \otimes \lambda_3)$), and when these hold $(\lambda_1 \otimes \lambda_2) \otimes \lambda_3 = \lambda_1 \otimes (\lambda_2 \otimes \lambda_3)$.

It is also obvious that $\delta \otimes \lambda = \lambda \otimes \delta = \lambda$ always holds, so the idle step is a unit for concurrent composition.

**Examples**

- Reads and writes with compatible start states can be executed in parallel, and reads and writes to distinct variables $x$ and $y$ never conflict:

$$\begin{align*}
\text{write}(x, v_1, v'_1) \otimes \text{write}(y, v_2, v'_2) &= ([x : v_1, y : v_2], [x : v'_1, y : v'_2]) \\
\text{read}(x, v_1) \otimes \text{read}(y, v_2) &= ([x : v_1, y : v_2], []) \\
\text{read}(x, v_1) \otimes \text{write}(y, v_2, v'_2) &= ([x : v_1, y : v_2], [y : v'_2]) \\
\text{read}(x, v_1) \otimes \text{read}(y, v_2) &= ([x : v_1, y : v_2], [])
\end{align*}$$

- For steps affecting the same single variable, concurrent reads are benign but concurrent writes constitute a race:

$$\begin{align*}
\text{read}(x, v) \otimes \text{read}(x, v) &= \text{read}(x, v) = ([x : v], []) \\
\text{write}(x, v, v') \otimes \text{read}(x, v) &= \text{write}(x, v, v') = ([x : v], [x : v']) \\
\text{write}(x, v, v'_1) \otimes \text{write}(x, v, v'_2) &= ([x : v], [x : \top])
\end{align*}$$

- A write to $x$ concurrent with a read of $x$ only constitutes a significant race if the read influences the value of some identifier. The following examples show how races get handled, specifically that the correct identifiers get tainted, when the flow relations are non-trivial. To facilitate comparison with the above definition we enumerate the flow relations explicitly.

(i) Consider a step that updates $y$ and reads $x$ but does not use the value of $x$ in the update. If we concurrently write to $x$, the combined effect is to update both
Brookes

\(x\) and \(y\) as intended. For example,

\[
([x : 1, y : 0], \{(y, y)\}, [y : 1]) \otimes ([x : 1], \{(x, x)\}, [x : 2])
\]

\[
= ([x : 1, y : 0], \{(x, x), (y, y), [x : 2, y : 1]\})
\]

(ii) Now consider a step that updates \(y\) using a value obtained by reading \(x\). If we concurrently write to \(x\) the value of \(y\) is tainted. For example,

\[
([x : 1, y : 0], \{(x, y), (y, y)\}, [y : 1]) \otimes ([x : 1], \{(x, x)\}, [x : 2])
\]

\[
= ([x : 1, y : 0], \{(x, x), (y, y)\}, [x : 2, y : \top])
\]

In each case the combined step accurately reflects the overall effect.

There is an obvious connection between concurrent composition and enabling.

**Theorem 14**

When \(\lambda\) and \(\mu\) have identity flow relations, \(\lambda \boweq \mu\), and \(\sigma \boweq \lambda \otimes \mu\), there are states \(\sigma_1, \sigma_2\) such that \(\sigma \Rightarrow \sigma_1, \sigma \Rightarrow \sigma_2\), and \(\sigma'\) is uniquely determined by the following properties:

- \(\sigma'(i) = \sigma_1(i)\) for \(i \in \text{writes}(\lambda) - \text{reads}(\mu)\)
- \(\sigma'(i) = \sigma_2(i)\) for \(i \in \text{writes}(\mu) - \text{reads}(\lambda)\)
- \(\sigma'(i) = 1\) for \(i \in \text{writes}(\lambda) \cap \text{reads}(\mu)\) or \(\text{writes}(\mu) \cap \text{reads}(\lambda)\)
- \(\sigma'(i) = \sigma(i)\) for \(i \in \text{dom}(\sigma) - (\text{writes}(\lambda) \cup \text{writes}(\mu))\).

A more general relationship is derivable, involving \(R\) and \(S\) in a natural manner.

**Traces**

Traces are finite sequences of steps, where each step is either an atomic resource action or represents the cumulative effect of a finite computation. We want to abstract away from the intermediate states in between resource actions, in the spirit of Dijkstra’s Principle: *processes should be regarded as independent, except when they synchronize* [5]. The basic information about reads, writes and dependencies contained in a step is sufficient for this purpose, and we can work with traces in which adjacent resource-free steps are consecutively executable and mumbled together. Thus we abstract away from the order in which intermediate reads and writes occur, since the mumbled step only reports the “initial” reads and the “final” writes. In doing this we lose no generality when considering the effect on the shared state as viewed by other processes running concurrently. The only way another process could be influenced by or affect an intermediate stage would be by reading or writing to a variable written by this step, causing a race condition. The mumbled step also writes to this variable, so we would also get a race condition under these circumstances and we lose no generality by keeping only the mumbled step.

To build traces with this reduced structure we introduce a modified form of concatenation, a partial operation \(\cdot\) that combines traces whose concatenation can
be properly reduced by such mumbling, and implements this reduction.

**Definition 15** \( \lambda \) precedes \( \mu \), written \( \lambda < \mu \), if \( \lambda \vdash \mu \) or \( \text{res}(\lambda) \cup \text{res}(\mu) \neq \{\} \). When this holds we define \( \lambda \cdot \mu \) to be the step given by:

\[
\lambda \cdot \mu = \lambda \mu \quad \text{if} \quad \text{res}(\lambda) \cup \text{res}(\mu) \neq \{\}
\]

\[
= \lambda \; \mu \quad \text{if} \quad \text{res}(\lambda) = \text{res}(\mu) = \{\} \quad \text{and} \quad \lambda \vdash \mu
\]

We refer to this operation from now on simply as concatenation. We extend to finite traces in the obvious inductive manner. Let \( \epsilon \) be the empty trace.

**Definition 16** For all \( \alpha, \beta, \gamma, \lambda, \mu \) we have \( \epsilon < \beta, \alpha < \epsilon, \) and \( (\alpha\lambda) < (\mu\beta) \) iff \( \lambda < \mu \). Further, \( \epsilon \cdot \beta = \beta, \alpha \cdot \epsilon = \alpha, \) and when \( \lambda < \mu \) we let \( (\alpha\lambda) \cdot (\mu\beta) = \alpha(\lambda \cdot \mu)\beta. \)

It is easy to verify that \( \delta \cdot \delta = \delta \) and that \( \cdot \) is associative.

**Theorem 17** For all traces \( \alpha, \beta, \gamma, \alpha < \beta \& (\alpha \cdot \beta) < \gamma \iff \beta < \gamma \& \alpha < (\beta \cdot \gamma) \) and when these hold, \( (\alpha \cdot \beta) \cdot \gamma = \alpha \cdot (\beta \cdot \gamma). \)

**Examples**

Clearly \( \text{write}(x, 0, 1) \cdot \text{write}(x, 1, 2) = \text{write}(x, 0, 2) \), and

\[
\text{lock}(r) \cdot \text{write}(x, 0, 1) = \text{lock}(r) \text{write}(x, 0, 1)
\]

\[
\text{lock}(r) \cdot \text{write}(x, 0, 1) \cdot \text{write}(x, 1, 2) \cdot \text{unlock}(r) = \text{lock}(r) \text{write}(x, 0, 2) \text{unlock}(r)
\]

**Reducing traces**

We say that a trace \( \alpha \) is reduced (or “mumbled”) iff for all pairs of successive steps \( \lambda\mu \) in \( \alpha \) either \( \text{res}(\lambda) \neq \{\} \) or \( \text{res}(\mu) \neq \{\} \). It is easy to see that any trace built using \( \cdot \) is reduced, because when \( \alpha\lambda \) and \( \mu\beta \) are reduced traces and \( \lambda < \mu \), \( \alpha(\lambda \cdot \mu)\beta \) is also a reduced trace. We say that \( \lambda_1 \ldots \lambda_n \) is feasible iff its steps can be combined using \( \cdot \), and when this happens we obtain the reduced trace \( \text{red}(\alpha) = \lambda_1 \ldots \lambda_n. \)

For example, the trace

\[
\alpha = \text{lock}(r) \text{write}(x, 0, 1) \text{write}(x, 1, 2) \text{unlock}(r) \text{lock}(r) \text{write}(x, 0, 1) \text{unlock}(r)
\]

is feasible, and \( \text{red}(\alpha) \) is the trace

\[
\text{lock}(r) \text{write}(x, 0, 2) \text{unlock}(r) \text{lock}(r) \text{write}(x, 0, 1) \text{unlock}(r).
\]

However the trace \( \text{write}(x, 0, 2) \text{write}(x, 0, 1) \) is not feasible.

Every reduced trace \( \text{write}(x, 0, 2) \text{write}(x, 0, 1) \) is also feasible and satisfies the equation \( \text{red}(\alpha) = \alpha. \)

Whenever \( \alpha \) is feasible, \( \text{red}(\alpha) \) is a reduced trace.

**Executing traces**

We extend the effect relation to traces in the obvious way:

\[
\sigma \xrightarrow{\lambda_1 \ldots \lambda_n} \sigma' \iff \exists \sigma_0, \ldots, \sigma_n. \sigma = \sigma_0 & \sigma_n = \sigma' \\
& \sigma_0 \xrightarrow{\lambda_1} \sigma_1 \xrightarrow{\lambda_2} \sigma_2 \ldots \sigma_{n-1} \xrightarrow{\lambda_n} \sigma_n
\]
When \( n = 0 \) we get \( \sigma \overset{\alpha}{\Rightarrow} \sigma \). When \( \sigma \overset{\alpha}{\Rightarrow} \sigma' \) we say that \( \alpha \) is enabled in \( \sigma \), and \( \sigma' \) is the result of executing \( \alpha \) from \( \sigma' \). A trace is \emph{executable} if it is enabled by some state.

We are mainly interested in executable traces, yet our semantics deals with \emph{feasible} traces, including both executable and non-executable traces. This is necessary because a parallel program may have an executable trace that arises from non-executable traces of its constituent processes. For example, the traces \( \alpha = \text{lock}(r) \text{lock}(r) \) and \( \beta = \text{unlock}(r) \text{unlock}(r) \) are both non-executable, but they have an interleaving of the form \( \text{lock}(r) \text{unlock}(r) \text{lock}(r) \text{unlock}(r) \), which is executable.

The executable traces of \( c_1 \parallel c_2 \) can be determined from the feasible traces of \( c_1 \) and \( c_2 \). Executable traces are also feasible, but the converse fails. For example, the trace

\[
\text{lock}(r) \text{write}(x, 0, 1) \text{unlock}(r) \text{lock}(r) \text{write}(x, 0, 1) \text{unlock}(r)
\]

is feasible but not executable. The claim that we lose no generality by dropping infeasible traces is validated by the facts that: when \( \alpha \) is infeasible there are no states \( \sigma \) such that \( \sigma \overset{\alpha}{\Rightarrow} \sigma' \), and there is no way to fill the gaps in \( \alpha \) without concurrently writing to a variable read by \( \alpha \).

**Parallel composition**

We adapt the definition of resource-sensitive fair merge from our prior work, adjusted to generate mumbled traces. For traces \( \alpha \) and \( \beta \), and disjoint finite sets of resources \( A \) and \( B \), we define the set of feasible merges

\[
\alpha_A \parallel_B \beta \subseteq \Lambda^*
\]

by induction on trace length. This set consists of all traces in which a process holding resources \( A \) runs concurrently with a process holding resources \( B \). We first define the relation \( A \overset{\lambda}{\rightarrow}_B A' \) that characterizes when the process holding \( A \) can do step \( \lambda \):

\[
\begin{align*}
A \overset{\text{lock}(r)}{\rightarrow}_B A \cup \{r\} & \quad \text{if } r \not\in A \cup B \\
A \overset{\text{unlock}(r)}{\rightarrow}_B A \setminus \{r\} & \quad \text{if } r \in A \\
A \overset{\lambda}{\rightarrow}_B A' & \quad \text{if } \text{res}(\lambda) = \{} 
\end{align*}
\]

When one (or both) of the traces is empty we define:

\[
\begin{align*}
\alpha_A \parallel_B \epsilon & = \{ \alpha \mid A \overset{\alpha}{\rightarrow}_B A' \} \\
\epsilon_A \parallel_B \beta & = \{ \beta \mid B \overset{\alpha}{\rightarrow}_A B' \}
\end{align*}
\]

For the inductive case we define:

\[
(\lambda \alpha)_A \parallel_B (\mu \beta) = \{ \lambda \cdot \gamma \mid A \overset{\lambda}{\rightarrow}_B A' \land \gamma \in \alpha_A \parallel_B (\mu \beta) \land \lambda \cdot \gamma \} \\
\cup \{ \mu \cdot \gamma \mid B \overset{\mu}{\rightarrow}_A B' \land \gamma \in (\lambda \alpha)_A \parallel_B \beta \land \mu \cdot \gamma \} \\
\cup \{ (\lambda \otimes \mu) \cdot \gamma \mid \lambda \text{ co } \mu, \gamma \in \alpha_A \parallel_B \beta \land (\lambda \otimes \mu) \cdot \gamma \}
\]

83
When \( A = B = \{\} \) we omit the subscripts and write \( \alpha \| \beta \).

The first two terms produce interleavings in which one process does a step first; the third term allows concurrent combinations when enabled, and takes account of the potential for race conditions. Use of \( \cdot \) ensures that we only include feasible traces. The reader can check that when \( \alpha \) and \( \beta \) are feasible traces, every trace belonging to \( \alpha \| \beta \) is also feasible.

When \( \lambda \) and \( \mu \) are concurrently enabled resource-free steps, \( \lambda \| \mu = \{\lambda \otimes \mu\} \). If in addition the steps are conflict-free, we have \( \lambda; \mu = \mu; \lambda = \lambda \otimes \mu \).

When \( \lambda \) and \( \mu \) are resource-free steps that are not consecutively executable in either order, and not concurrently executable, \( \lambda \| \mu = \{\} \).

When at least one of \( \lambda \) and \( \mu \) is a resource step, \( \lambda \| \mu = \{\lambda \mu, \mu \lambda\} \).

Examples

• The following examples check that the above definitions conform with intuition in cases involving concurrent writes. In each case we obtain the set of all (reduced) feasible combinations. Note that several distinct interleavings may lead to the same reduced trace, showing the succinctness of our construction.

(i) Concurrent writes to distinct variables: \( \text{write}(x, 0, 1) \| \text{write}(y, 0, 1) \) yields three feasible interleavings:

\[
\begin{align*}
\text{write}(x, 0, 1) \cdot \text{write}(y, 0, 1) \\
\text{write}(y, 0, 1) \cdot \text{write}(x, 0, 1) \\
\text{write}(x, 0, 1) \otimes \text{write}(y, 0, 1)
\end{align*}
\]

and each of these reduces to the same trace \( ([x : 0, y : 0], [x : 1, y : 1]) \). So \( \text{write}(x, 0, 1) \| \text{write}(y, 0, 1) = \{([x : 0, y : 0], [x : 1, y : 1])\} \).

(ii) Concurrently executable writes to the same variable:

\[
\begin{align*}
\text{write}(x, 0, 1) \| \text{write}(x, 0, 1) \\
&= \{\text{write}(x, 0, 1) \otimes \text{write}(x, 0, 1)\} \\
&= \{([x : 0], [x : \top])\}.
\end{align*}
\]

In this case the updates are not sequentially composable.

(iii) Non-concurrently executable writes to the same variable:

\[
\begin{align*}
\text{write}(x, 0, 1) \| \text{write}(x, 1, 2) \\
&= \{\text{write}(x, 0, 1) \cdot \text{write}(x, 1, 2)\} \\
&= \{\text{write}(x, 0, 2)\}
\end{align*}
\]

because there is only one feasible interleaving. On the other hand, \( \text{write}(x, 0, 1) \| \text{write}(x, 2, 1) = \{\} \), because the two steps cannot be composed either sequentially or concurrently.

• Let \( \alpha \) be the trace \( \text{lock}(r) \text{write}(x, 0, 1) \text{unlock}(r) \). The trace set \( \alpha \| \alpha \) is \( \{\alpha \alpha\} \).

Other interleavings begin with consecutive \( \text{lock}(r) \) steps and fall afoul of the re-
source constraints built into the interleaving operator; such traces are irrelevant in any case because none are executable, and none can be used to construct an executable trace by interleaving. Similarly, let $\beta$ be $\text{lock}(r) \text{write}(x,1,0) \text{unlock}(r)$. The set $\alpha \parallel \beta$ is just $\{\alpha\beta, \beta\alpha\}$. One interleaving of the (non-executable) trace $\alpha\alpha$ with $\beta$ is the executable trace $\alpha\beta\alpha$.

**Semantics of expressions**

Since our semantics is designed to detect race conditions involving concurrent access to shared variables, we can work with a very simple model for expressions. There is no need to keep track of the order in which reads occur during expression evaluation, provided we record the set of reads on which the expression value depends, so we can use a set of state-value pairs. For an integer expression $e$ we will let $[e] \subseteq \Sigma \times V$, and for a boolean expression $b$ we let $[b] \subseteq \Sigma \times \{\text{true}, \text{false}\}$. An entry $(\sigma, v) \in [e]$ represents the fact that evaluation of $e$ in a $\tau$ such that $\tau \supseteq \sigma$ only reads the portion $\sigma$ of the state and produces the integer value $v$, and similarly for boolean expressions. So each entry is a minimal piece of computational information about expression evaluation, in line with our desire to deal with footprints. We define $[e]$ and $[b]$ by structural induction as usual:

$$[n] = \{(\text{[ ]}, n)\}$$
$$[i] = \{(i : v), v \in V\}$$
$$[e_1 + e_2] = \{(\sigma_1 \cup \sigma_2, v_1 + v_2) \mid (\sigma_1, v_1) \in [e_1] \& (\sigma_2, v_2) \in [e_2] \& \sigma_1 \uparrow \sigma_2\}$$
$$[\text{true}] = \{([\text{ ]}, \text{true})\}$$
$$[e_1 = e_2] = \{(\sigma_1 \cup \sigma_2, v_1 = v_2) \mid (\sigma_1, v_1) \in [e_1] \& (\sigma_2, v_2) \in [e_2] \& \sigma_1 \uparrow \sigma_2\}$$

We remark that when $(\sigma, v) \in [e]$ the piece of state $\sigma$ contains values for the identifiers occurring free in $e$ whose values are used to compute $v$.

**Theorem 18** For all expressions $e$, if $(\sigma, v) \in [e]$ then $\text{dom}(\sigma) \subseteq \text{free}(e)$.

**Semantics of commands**

Commands denote sets of feasible traces. We define the trace set of a command, $[c] \subseteq \mathcal{P}(\Lambda^*)$, by structural induction. To simplify the presentation we introduce
the abbreviations $[b]_{true}$ for $\{(\sigma, []) \mid (\sigma, true) \in [b]\}$ and similarly for $[b]_{false}$.

\[
\begin{align*}
[\text{skip}] & = \{\delta\} \\
[i:=e] & = \{((\sigma \cup [i : v], U, [i : v']) \mid (\sigma, v') \in [e] \& \sigma \uparrow [i : v]) \} \\
& \quad \text{where } U^{-1}(i) = dom(\sigma) \cup \{i\} \\
[c_1;c_2] & = [c_1] \cdot [c_2] = \{\alpha_1 \cdot \alpha_2 \mid \alpha_1 \in [c_1], \alpha_2 \in [c_2] \& \alpha_1 \triangleleft \alpha_2\} \\
[\text{if } b \text{ then } c_1 \text{ else } c_2] & = [b]_{true} \cdot [c_1] \cup [b]_{false} \cdot [c_2] \\
[\text{while } b \text{ do } c] & = ([b]_{true} \cdot [c])^* \cdot [b]_{false} \\
[c_1||c_2] & = [c_1]| [c_2] = \bigcup\{\alpha_1||\alpha_2 \mid \alpha_1 \in [c_1] \& \alpha_2 \in [c_2]\} \\
[\text{with } r \text{ when } b \text{ do } c] & = \{\text{lock}(r) \cdot \beta \cdot \gamma \text{ unlock}(r) \mid \beta \in [b]_{true}, \gamma \in [e], \beta \triangleleft \gamma\}
\end{align*}
\]

Note the careful use of $\triangleleft$ and $\cdot$ (crucially in the semantic clauses for sequential and parallel composition, but throughout the above presentation) to ensure that the trace set $[c]$ consists of feasible traces. Also note the implicit use of the sequencing and concurrent composition operations on steps, to reduce traces and abstract away from the order of action occurrences in between resource steps. The executable traces of $c$ can be extracted from $[c]$ as a subset.

The semantic clause for critical regions shows how entry to and exit from a region are modeled as lock($r$) and unlock($r$), which together with the way we defined resource-sensitive interleaving ensures that the semantic clause for $c_1||c_2$ correctly models concurrent execution while obeying the atomicity and mutual exclusion constraints on resources. Since we only care here about partial correctness behavior there is no need to include infinite traces and no need to include busy waiting traces caused by unavailable resources and/or the falsity of the entry condition $b$. Similarly the clause for while-loops does not include infinite iteration.

The semantic clause for assignment conceals a slight subtlety: commands do not allocate storage during execution, so an assignment can only be executed from a state in which it target variable already has a value. So the footprint of $i:=e$ needs a read state in which $i$ has a value and from which $e$ can be evaluated. To allow for cases where $i$ is not relevant to the value of $e$ as well as for cases where $e$ needs the value of $i$, we use $\sigma \cup [i : v]$ (with $\sigma \uparrow [i : v]$) as the read state in the footprint of $i:=e$, where $(\sigma, v) \in [e]$). When $i \notin dom(\sigma)$, so that $i$ is not needed by $e$, the consistency constraint holds for all $v$; when $i \in dom(\sigma)$, so the value of $e$ may depend on $i$, the consistency constraint only holds for $v = \sigma(i)$ and the state $\sigma \cup [i : v]$ is the same as $\sigma$. 86
Examples

(i) We illustrate how our semantics ignores the relative order of individual assignments and takes account of race conditions:

\[ [x:=1] = \{ ([x:v], [x:1]) \mid v \in V \} \]
\[ [x:=1; x:=2] = [x:=2] = \{ ([x:v], [x:2]) \mid v \in V \} \]
\[ [x:=1; y:=1] = [x:=1||y:=1] = \{ ([x:v_1, y:v_2], [x:1, y:1]) \mid v_1, v_2 \in V \} \]
\[ [x:=1||x:=1] = \{ ([x:v], [x:T]) \mid v \in V \} \]

(ii) The next example illustrates the relevance of flow relations. It is easy to see that \([x:=1; y:=x] = \{ ([x:v_1, y:v_2], R, [x:1, y:1]) \mid v_1, v_2 \in V \} \), where \( R = \{ (x,x), (y,y), (x,y) \} \). When we run this command in parallel with \( x:=2 \) there is a race condition involving \( x \) and \( y \):

\[ \{ [x:=1; y:=x]|y:=2] = \{ ([x:v_1, y:v_2], [x:T, y:T]) \mid v_1, v_2 \in V \}. \]

Our semantics does distinguish between \( x:=1; y:=x \) and \( x:=1; y:=1 \), and this is necessary because in the latter command the value of \( y \) is not influenced by the value of \( x \), leading to different behavior in contexts that write to \( x \). Indeed, in contrast to the previous example, we have

\[ \{ [x:=1; y:=1]|y:=2] = \{ ([x:v_1, y:v_2], [x:T, y:1]) \mid v_1, v_2 \in V \}. \]

(iii) A slightly more complex example shows how expression evaluation and control flow fit in: \[ \text{if } x > 0 \text{ then } y:=1 \text{ else } z:=1 \] is the set

\[ \{ ([x:v_1, y:v_2], U_y, [y:1]) \mid v_1 > 0, v_2 \in V \} \]
\[ \cup \{ ([x:v_1, z:v_2], U_z, [z:1]) \mid v_1 \leq 0, v_2 \in V \} \]

where \( U_y^{-1}(y) = \{ x, y \} \) and \( U_z^{-1}(z) = \{ x, z \} \).

(iv) For a while-loop with no critical regions, our semantics abstracts away from the intermediate states generated by successive iterations:

\[ [\text{while } y > 0 \text{ do } (x:=x+1; y:=y-1)] \]
\[ = \{ ([x:v_1, y:v_2], U, [x:v_1+v_2, y:0]) \mid v_2 > 0 \} \cup \{ ([y:v], []) \mid v \leq 0 \} \]

where \( U^{-1}(x) = \{ x, y \} \).

(v) The only traces of the program \((\text{with } r \text{ do } x:=1)|| (\text{with } r \text{ do } x:=2)\) that are executable from a state in which the values of \( r \) and \( x \) are 0 are

\[ \text{lock}(r) \text{ write}(x, 0, 1) \text{ unlock}(r) \text{ lock}(r) \text{ write}(x, 1, 2) \text{ unlock}(r) \]

and \[ \text{lock}(r) \text{ write}(x, 0, 2) \text{ unlock}(r) \text{ lock}(r) \text{ write}(x, 2, 1) \text{ unlock}(r). \]

(vi) Consider the program \( (x:=1; x:=2)|| \text{with } r \text{ do } y:=x \). Each of its traces arises by interleaving a trace \([x:v],[x:2]\) of \( x:=1; x:=2 \) with a trace of form
Consider the program with \( \tau, R, \tau' \) singletons consisting of single step \( c \). If in addition to this \( R \) is \( \{ [x] \} \), and \( \sigma \) is \( \{ \alpha(\sigma, R, \sigma') \} \). For example, \( \text{Example 3.1} \)

Suppose \( x \neq \text{free}(e_2) \) and \( y \neq \text{free}(e_1) \). Then

\[
[x := e_1; y := e_2] = \{ (\sigma_1 \cup \sigma_2, R, [x : v_1, y : v_2]) \mid (\sigma_1, v_1) \in [e_1], (\sigma_2, v_2) \in [e_2], \sigma_1 \uplus \sigma_2, \]

\[
R^{-1}(x) = \text{dom}(\sigma_1) \& R^{-1}(y) = \text{dom}(\sigma_2) \}
\]

It follows that \([x := e_1; y := e_2] = [y := e_2; x := e_1] = [x := e_1 || y := e_2] \].

Recall that a state \( \sigma \) is race-free iff \( \top \neq \text{rgf}(\sigma) \).

**Definition 20** \( c \) is race-free from \( \sigma \) iff \( \forall \alpha \in [c]. \forall \sigma' \in [\sigma]. \sigma \Rightarrow \sigma' \) implies \( \sigma' \) race-free.

When this holds for all states \( \sigma \) we say that \( c \) is race-free.

**Theorem 21**

- \( \text{skip} \) and \( \text{i:=c} \) are race-free.
• If \( c_1 \) and \( c_2 \) are race-free, so are \( c_1; c_2 \) and \( \text{if } b \text{ then } c_1 \text{ else } c_2 \).
• If \( c \) is race-free, so are \( \text{while } b \text{ do } c \) and \( \text{with } r \text{ when } b \text{ do } c \).
• If \( \text{writes}(c_1) \cap \text{free}(c_2) = \text{writes}(c_2) \cap \text{free}(c_1) = \{\} \), and \( c_1 \) and \( c_2 \) are race-free, then \( c_1 || c_2 \) is race-free.

**Corollary** Every sequential program \( c \) is race-free.

A resource context \( \Gamma \) is a set of entries of form \( r(X) \), where \( r \) is a resource name and \( X \) is a finite set of identifiers, called a protection list. We say that \( c \) respects \( \Gamma \) iff every free occurrence in \( c \) of an identifier protected by \( r \) in \( \Gamma \) is inside a conditional critical region naming \( r \). The next result shows that our semantics validates the Owicki-Gries static constraints on critical variables [9].

**Theorem 22**
Let \( c_1 \) and \( c_2 \) be race-free commands. If \( c_1 \) and \( c_2 \) respect \( \Gamma \) and

\[
\text{writes}(c_1) \cap \text{free}(c_2) \subseteq \text{owned}(\Gamma) \land \text{writes}(c_2) \cap \text{free}(c_1) \subseteq \text{owned}(\Gamma)
\]

then \( c_1 || c_2 \) is race-free.

**Safe partial correctness**

Let \( p \) and \( q \) be boolean-valued expressions in which identifiers may occur free. (We do not allow resource names to appear free in such expressions.) Let \(|p|\) be the set of states satisfying \( p \).

The *safe partial correctness* formula \( \{p\}c\{q\} \) is valid iff all finite executions of \( c \) from a state satisfying \( p \) are race-free and end in a state satisfying \( q \).

**Definition 23** \( \{p\}c\{q\} \) is valid for all \( \alpha \in |c| \), and all states \( \sigma \) such that \( \text{dom}(\sigma) \supseteq \text{free}(p, c, q) \), if \( \sigma \in |p| \) and \( \sigma \xrightarrow{\sigma'} \sigma' \), then \( \sigma' \) is race-free and \( \sigma' \in |q| \).

**Theorem 24** If \([c_1] = [c_2]\) then \( c_1 \) and \( c_2 \) satisfy the same assertions in all program contexts, i.e. for all \( p \) and \( q \), and all contexts \( C[-] \), \( \{p\}C[c_1]\{q\} \) is valid iff \( \{p\}C[c_2]\{q\} \) is valid.

**Corollary 25** If \([c_1] = [c_2]\) then \( c_1 \) and \( c_2 \) satisfy the same assertions, i.e. for all \( p \) and \( q \), \( \{p\}c_1\{q\} \) is valid iff \( \{p\}c_2\{q\} \) is valid.

**Observing the state**

For a trace \( \alpha \), let \(|\alpha|\) be the set of state sequences obtainable by executing \( \alpha \). When \( \alpha = \lambda_1 \ldots \lambda_n \) we have \(|\alpha| = \{\sigma_1 \ldots \sigma_n \mid \sigma_1 \xrightarrow{\lambda_1} \sigma_2 \ldots \sigma_{n-1} \xrightarrow{\lambda_n} \sigma_n\} \). When \( \alpha \) is non-executable this set is obviously empty. We define the set \( S(c) \) of state sequences observable during executions of \( c \), to be \( S(c) = \bigcup\{|\alpha| \mid \alpha \in |c|\} \). This is observing the state at start and finish and at all synchronization points, since we are dealing here with mumbled traces.

**Theorem 26** If \([c_1] = [c_2]\) then for all contexts \( C[-] \), \( S(C[c_1]) = S(C[c_2]) \).

89
4 Conclusions

Our more refined treatment of races leads to a footprint trace semantics that supports compositional reasoning about safe partial correctness, and accurately tracks the variables whose values are immune to race conditions. The new model enjoys better abstraction properties than the earlier version of footprint semantics [4]. We can pinpoint the differences by looking at the notions of semantic equivalence induced by the old semantics and the new. In the earlier model a race condition was interpreted as a global catastrophe, leading to a special abort state. This earlier model gave the same meaning, namely

$$\{([x:v_1, y:v_2], [x:1, y:1]) \mid v_1, v_2 \in V\},$$

to (i) $x:=1; y:=x$ and (ii) $x:=1; y:=1$, and hence the same meaning, namely

$$\{([x:v_1, y:v_2], \text{abort}) \mid v_1, v_2 \in V\},$$

to programs (i)$\|x:=2$ and (ii)$\|x:=2$. The new semantics distinguishes between (i) and (ii), because they induce different flow relations: only in (i) is the value of $y$ dependent on $x$. (We made a similar comment in an example, earlier.) Moreover, this distinction is worth making, because when run in parallel with a program that (only) writes to $x$, for (i) the value of $y$ gets tainted but for (ii) it does not. Thus there is a program context in which (i) and (ii) have different observable behavior. In particular, the new semantics distinguishes (correctly) between the programs (i)$\|x:=2$ and (ii)$\|x:=2$.

We refer to our semantics as “grainless”. We argue that this appellation is accurate, because the steps employed in the traces of programs are obtained by “mumbling” consecutive actions together in between resource actions, so that we abstract away from irrelevant details concerning scheduling and atomicity. We do still assume that resource actions lock($r$) and unlock($r$) are implemented atomically. Further, we only include feasible traces, leading to a more succinct trace set that subscribes roughly speaking to the slogan: fewer traces, shorter traces, bigger steps. Thus our semantics strives to avoid the combinatorial explosion inherent in interleaving traces, by working with smaller trace sets and shorter traces. Our more localized account of races allows a more liberal view of safe partial correctness. We could say that $\{p\}c\{q\}$ is valid iff for all states $\sigma$ satisfying $p$, and all $\alpha \in [c]$, if $\sigma \xrightarrow{\alpha} \sigma'$ then $\sigma'$ satisfies $q$. Assuming that we define satisfaction so that $\sigma \in [p]$ implies $\forall i \in \text{free}(p)$. $\sigma(i) \neq \top$, this notion of validity does not imply absolute race-freedom of $c$ from states satisfying the pre-condition, just that the state relevant to the post-condition is race-free. We plan to explore this idea further, perhaps leading to a new variant of concurrent separation logic that deals more flexibly with race conditions.

We dealt here with a simple shared-memory parallel language: no pointers or mutable state, no dynamic allocation of storage. We believe that the foundations laid in our prior development of action trace semantics for concurrent separation
logic [3] and our earlier grainless model [4] can be adapted to combine the new approach to race-detection and tainting with mutable state, and we plan to investigate further. We also intend to explore existing work using flow analysis in other settings, such as secure information flow and program analysis. Insights from related fields may help us to assess the scope and limitations of our work, and may suggest some improvements.

We have so far assumed that programs are executed on an architecture that provides sequential memory consistency [7]:

... the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.

This assumption leads naturally to the use of interleaving to represent parallel composition, a common feature in denotational models of shared-memory dating back to Park [10]. We have not yet applied our semantic framework to deal with weaker memory assumptions, such as those actually supported by some modern concurrency architectures [1]; it would be interesting to see to what extent our semantics can help support reasoning about language implementations that offer weaker guarantees about the observable effect of concurrent updates. In any case we would argue that even in (perhaps especially in) dealing with more relaxed implementations it would be useful to have a semantics like ours, that yields a (machine-independent) characterization of race-free programs and predicts (semantically) to what extent a program’s behavior is susceptible to uncertainty about the values of variables.

Acknowledgements

We thank the referees for some helpful suggestions and pointers to potentially interesting related work.

References


Strategies as Concurrent Processes

Simon Castellan\textsuperscript{a}  Jonathan Hayman\textsuperscript{a}  Marc Lasson\textsuperscript{b}  Glynn Winskel\textsuperscript{a}

\textsuperscript{a} Computer Laboratory, University of Cambridge, UK
\textsuperscript{b} INRIA Paris-Rocquencourt, PiR2, Univ Paris Diderot, Sorbonne Paris Cité F-78153, Le Chesnay, France

Abstract
Concurrent strategies are shown to support operations yielding an economic yet rich higher-order concurrent process language, which shares features both with process calculi and nondeterministic dataflow. Its operational semantics and ‘may and must’ equivalence require that we take internal (neutral) moves seriously, leading to the introduction of ‘partial strategies’ which may contain neutral moves. Through partial strategies, we can present a transition semantics for a language of strategies and can formulate their ‘may and must’ behaviour. While partial strategies compose, in a way extending that of strategies, in general composition introduces extra neutral moves; in particular, copy-cat is no longer strictly an identity w.r.t. composition. However, a simple extension of concurrent strategies (with stopping configurations) maintains the fact that they form a bicategory while still capturing ‘may and must’ behaviour.

1 Introduction
There are several reasons for extending games and strategies, with behaviour based on trees, to concurrent games and strategies, based on event structures — the concurrent analogue of trees. One reason is to provide a foundation for a generalized domain theory, in which concurrent games and strategies take over the roles of domains and continuous functions. The motivation is to repair the divide between denotational and operational semantics and tackle anomalies like nondeterministic dataflow, which are beyond traditional domain theory. Another is that strategies are as potentially fundamental as relations and functions. It is surely because of our limited mental capacity, and not because of its unimportance, that the mathematical concept of strategy has been uncovered relatively late. It is hard to think about the successive contingencies involved in playing a game in the same way that is hard to think about interacting processes. Developing strategies in the extra generality demanded by concurrency reveals more clearly their essential nature and enables us to harness computer-science expertise in structure and concurrency in their understanding and formalization.
The extra generality of concurrency reveals new structure and a mathematical robustness to the concept of strategy, in particular showing strategies are essentially special profunctors [19]. Profunctors themselves provide a rich framework in which to generalize domain theory, in a way that is arguably closer to that initiated by Dana Scott than game semantics [9,2]. However, the mathematical abstraction of profunctors comes at a price: it can be hard to give an operational reading to denotations as profunctors. There are examples of semantics of higher-order process languages and “strong correspondence” where elements of profunctor denotations correspond to derivations in an operational semantics [11,15]. But in general it is hard to extract operational semantics from the profunctor denotations alone because they have abstracted too far.

Connections between forms of strategy and process models have been the subject of a large body of prior research — see e.g. [8,6,7]. In this paper, we begin a study of concurrent strategies from the perspective of concurrent processes, considering how concurrent games and strategies are objects which we can program. They are shown to support operations yielding an economic yet rich higher-order concurrent process language, which shares features both with process calculi and nondeterministic dataflow. The operations allow recursion to be interpreted using a trace and novel duplication operation.

Indeed, seen from a concurrent-process perspective, in some respects concurrent strategies have abstracted too far. Both in obtaining a transition semantics for the language of strategies and in analysing its behaviour w.r.t. ‘may and must’ testing, we need to take internal (neutral) moves, introduced in the composition of strategies, seriously. Through the more refined model of partial concurrent strategies we obtain a correspondence between events of a strategy and derivations of atomic steps in a transition semantics. Via partial strategies we can justify a simple extension of concurrent strategies (with stopping configurations) which maintains the fact that they form a bicategory, while still capturing ‘may and must’ behaviour.

2 Event structures and their maps

An event structure comprises \((E, \text{Con}, \leq)\), a set \(E\) of events which are partially ordered by \(\leq\), the causal dependency relation, and a nonempty consistency relation \(\text{Con}\) consisting of finite subsets of \(E\), which satisfy

\[
\{e' \mid e' \leq e\} \text{ is finite for all } e \in E \quad \{e\} \in \text{Con} \text{ for all } e \in E
\]

\[
Y \subseteq X \in \text{Con} \implies Y \in \text{Con} \quad X \in \text{Con} \& e \leq e' \in X \implies X \cup \{e\} \in \text{Con}.
\]

The configurations, \(C^\infty(E)\), of an event structure \(E\) consist of those subsets \(x \subseteq E\) which satisfy \(\forall X \subseteq x. \ X \text{ finite } \implies \ X \in \text{Con} \) (consistency) and \(\forall e, e'. \ e' \leq e \in x \implies e' \in x\) (down-closure). Often we shall be concerned with just the finite configurations of an event structure. We write \(C(E)\) for the finite configurations of an event structure \(E\).

We say an event structure is elementary when the consistency relation consists of all finite subsets of events. 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 $e \rightarrow e'$, meaning $e$ and $e'$ are distinct with
$e \preceq e'$ and no event in between, will play a very important role. For $X \subseteq E$ we
write $[X]$ for $\{e \in E \mid \exists e' \in X. e \preceq e'\}$, the down-closure of $X$; note if $X \in \text{Con}$, then
$[X] \in \text{Con}$. We write $[a]$ for $\{\{a\}\}$ where $a \in E$. For configurations $x, y$, we use
$x \prec y$ to mean $y$ covers $x$, i.e. $x \preceq y$ with nothing in between, and $x \prec e \preceq y$ to mean
$x \cup \{e\} = y$ for an event $e \notin x$. We sometimes use $x \preceq c$, expressing that event $e$
is enabled at configuration $x$, when $x \prec y$ for some configuration $y$.

Certain ‘structural’ maps of event structures, which have a long history [14], play
a key role in the development of nondeterministic concurrent strategies. A map
of event structures $f : E \rightarrow E'$ is a partial function $f : E \rightarrow E'$ such that $fx \in C^\infty(E')$
for all $x \in C^\infty(E)$, and for all $e_1, e_2 \in x$
$$f(e_1) = f(e_2) \& f(e_1), f(e_1) \text{ both defined } \implies e_1 = e_2.$$ 
Above, it is sufficient to restrict to finite configurations. Note that, when $f$ is total,
it restricts to a bijection $x \cong fx$ for any $x \in C^\infty(E)$. A total map is rigid when it
preserves causal dependency.

A map $f : E \rightarrow E'$ of event structures has partial-total factorization as a com-
position $E \xrightarrow{\pi_2} V \xrightarrow{(\varepsilon_{V},e)\text{ def }=\{e \in E \mid f(e) \text{ is defined}\}}$ is the domain
of $f$; the event structure $E \downarrow V = \text{def } (V, \leq_{V}, \text{Con}_{V})$, where $v \leq v'$ if $v \leq v'$ & $v, v' \in V$
and $X \in \text{Con}_{V}$ iff $X \in \text{Con} \& X \subseteq V$; the partial map $p : E \rightarrow E \downarrow V$
acts as identity on $V$ and is undefined otherwise; and the total map $t : E \downarrow V \rightarrow E'$,
called the defined part of $f$, acts as $f$. The event structure $E \downarrow V$ is the projection
of $E$ to $V$.

It shall be convenient to construct event structures using rigid families.

**Proposition 2.1** Let $Q$ be a non-empty family of finite partial orders closed under
rigid inclusions, i.e. if $q \in Q$ and $q' \overset{\rightarrow}{\to} q$ is a rigid inclusion (regarded as a map
of elementary event structures) then $q' \in Q$. The family $Q$ determines an event
structure $\text{Pr}(Q) = \text{def } (P, \leq, \text{Con})$ as follows:

- the events $P$ are primes, i.e. finite partial orders in $Q$ with a top element;
- the causal dependency relation $p' \preceq p$ holds precisely when there is a rigid inclusion
  from $p' \overset{\rightarrow}{\to} p$;
- a finite subset $X \subseteq P$ is consistent, $X \in \text{Con}$ iff there is $q \in Q$ and rigid inclusions
  $p \overset{\rightarrow}{\to} q$ for all $p \in X$.

If $x \in C(P)$ then $\cup x$, the union of the partial orders in $x$, is in $Q$. The function
$x \mapsto \cup x$ is an order-isomorphism from $C(P)$, ordered by inclusion, to $Q$, ordered by
rigid inclusions.

**Pullbacks of total maps** Maps $f : A \rightarrow C$ and $g : B \rightarrow C$ have
pullbacks in the category of event structures, and are simple to
describe in the case where $f$ and $g$ are total. In this situation,
finite configurations of $P$ correspond to the composite bijections
$$\theta : x \cong fx = gy \cong y$$
between configurations \( x \in \mathcal{C}(A) \) and \( y \in \mathcal{C}(B) \) s.t. \( fx = gy \) for which the transitive relation generated on \( \theta \) by \( (a, b) \leq (a', b') \) if \( a \leq_A a' \) or \( b \leq_B b' \) is a partial order; the correspondence taking \( z \in \mathcal{C}(P) \) to the composite bijection \( \pi_1 z \equiv f \pi_1 z = g \pi_2 z \equiv \pi_2 z \) respects inclusion.

2.1 Affine maps

In considering the dynamics of processes we shall need to relate a process to a process it may become. For this we generalize the earlier structural maps of event structures to affine maps, in which we need no longer preserve the empty configuration [16].

**Definition 2.2** Let \( A \) be an event structure. Let \( x \in \mathcal{C}^\infty(A) \). Write \( A/x \) for the event structure which remains after the occurrence of \( x \). Precisely, \( A/x \) comprises

- events, \( \{a \in A \setminus x \mid x \cup [a]_A \in \mathcal{C}^\infty(A)\} \),
- consistency relation, \( x \in \text{Con iff } X \subseteq \text{fin } A/x \& x \cup [X]_A \in \mathcal{C}^\infty(A) \), and
- causal dependency, the restriction of that on \( A \).

We extend the notation to configurations regarding them as elementary event structures. If \( y \in \mathcal{C}^\infty(A) \) with \( x \subseteq y \) then by \( y/x \) we mean the configuration \( y \setminus x \in \mathcal{C}^\infty(A/x) \). In the case of a singleton configuration \( \{a\} \) of \( A \) — when \( a \) is an initial event of \( A \) — we shall often write \( A/a \) and \( x/a \) instead of \( A/\{a\} \) and \( x/\{a\} \).

An affine map of event structures \( f \) from \( A \) to \( B \) comprises a pair \((f_0, f_1)\) where \( f_0 \in \mathcal{C}(B) \) and \( f_1 \) is a map of event structures \( f_1 : A \to B/f_0 \). It determines a function from \( \mathcal{C}(A) \) to \( \mathcal{C}(B) \) given by \( fx = f_0 \cup f_1 x \) for \( x \in \mathcal{C}(A) \). The allied \( f_0 \) and \( f_1 \) can be recovered from the action of \( f \) on configurations: \( f_0 = f\emptyset \) and \( f_1 \) is that unique map of event structures \( f_1 : A \to B/f\emptyset \) which on configurations \( x \in \mathcal{C}(A) \) returns \( fx/f\emptyset \). It is simplest to describe the composition \( gf \) of affine maps \( f = (f_0, f_1) \) from \( A \) to \( B \) and \( g = (g_0, g_1) \) from \( B \) to \( C \) in terms of its action on configurations: the composition takes a configuration \( x \in \mathcal{C}(A) \) to \( g(f x) \). Alternatively, the composition \( gf \) can be described as comprising \((g_0 \cup g_1 f_0, h)\) where \( h \) is that unique map of event structures \( h : A \to C/(g_0 \cup g_1 f_0) \) which sends \( x \in \mathcal{C}(A) \) to \( g_1(f_0 \cup f_1 x)/g_1 f_0 \). Note that traditional maps can be identified with those special affine maps \((f_0, f_1)\) in which \( f_0 = \emptyset \). We reserve the term ‘map’ for the traditional structural maps of event structure and shall say explicitly when a map is affine.

3 Concurrent games and strategies

A game is represented by an event structure \( A \) in which an event \( a \in A \) carries a polarity \( \text{pol}(a) \), + for Player and − for Opponent. Maps and affine maps of event structures extend to event structures with polarity: their underlying partial functions on events are required, where defined, to preserve polarity. A number of constructions on games will play an important role in the coming semantics:

**Dual** \( A^\perp \), of an event structure with polarity \( A \) is a copy of the event structure \( A \) with a reversal of polarities.

**Simple parallel composition** \( A \parallel B \), by juxtaposition. Its unit is the empty game.
\[ \emptyset. \] More generally, we can define, \( \parallel_{i \in I} A_i \), the simple parallel composition of a indexed family of games, in which the set of events comprises the disjoint union \( \bigcup_{1 \leq i \leq m} \{ i \} \times A_i \).

**Sums and recursive definitions** Sums \( \Sigma_{i \in I} A_i \) of an indexed family of games, which coincide with coproducts in the categories of event structures, are obtained in a similar way to simple parallel compositions, but now with events from distinct components being inconsistent (i.e. no set in the consistency relation contains elements from distinct components). We shall not make explicit use of recursively-defined games, but they are dealt with in exactly the same way as recursively-defined event structures \([14]\).

### 3.1 A bicategory of games and strategies

A (nondeterministic concurrent) strategy in a game \( A \) is represented by a total map \( \sigma : S \rightarrow A \) which preserves polarities and is

**Receptive:** if \( \sigma x \overset{\theta}{\rightarrow} c \) & \( \text{pol}_A(a) = - \) then there is unique \( s \in S \) s.t. \( x \overset{\theta}{\rightarrow} c \) & \( \sigma(s) = a \);

**Innocent:** if \( s \overset{\theta}{\rightarrow} s' \) & \( \text{pol}(s) = + \text{ or } \text{pol}(s') = - \) then \( \sigma(s) \overset{\theta}{\rightarrow} A \sigma(s') \).

Receptivity expresses that Player cannot hinder moves of Opponent, while innocence says a strategy should only adjoin immediate causal dependencies of the form \( \ominus \rightarrow \ominus \). A map between strategies from \( \sigma : S \rightarrow A \) to \( \sigma' : S' \rightarrow A \) is a total map \( f : S \rightarrow S' \) of event structures with polarity such that \( \sigma = \sigma' f \). Accordingly, the strategies are isomorphic iff \( f \) is an isomorphism of event structures.

The conditions of receptivity and innocence are necessary and sufficient to ensure that the copy-cat strategy behaves as identity w.r.t. composition \([12]\), which we now proceed to define.

We follow Conway and Joyal, and define a strategy from a game \( A \) to a game \( B \), written \( \sigma : A \rightarrow B \), as a strategy \( \sigma : A \rightarrow B \) in the game \( A \parallel B \).

Let \( \sigma : S \rightarrow A \parallel B \), \( \tau : T \rightarrow B \parallel C \) be strategies. Their composition is defined via the pullback drawn below. Ignoring polarities, the composite partial map has defined part \( T \parallel S \), which yields the composition of strategies \( \tau \circ \sigma : T \parallel S \rightarrow A \parallel C \) once polarities are reinstated.

\[
\begin{array}{ccc}
\pi_2 & A & \parallel T \\
& A \parallel \tau & \nearrow \\
S & B & \parallel C \\
\pi_1 & & \searrow \sigma \parallel C
\end{array}
\]

Let \( A \) be a game. The copy-cat strategy from \( A \) to \( A \) is a total map \( \gamma_A : \mathcal{C}_A \rightarrow A \parallel A \), based on the idea that Player moves, of +ve polarity, always copy previous corresponding moves of Opponent, of -ve polarity. For \( c \in A \parallel A \) we use \( \bar{c} \) to mean the corresponding copy of \( c \), of opposite polarity, in the alternative component. Define \( \mathcal{C}_A \) to comprise the event structure with polarity \( A \parallel A \) together with extra causal dependencies \( \bar{c} \leq \mathcal{C}_A c \) for all events \( c \) with \( \text{pol}_{A \parallel A}(c) = + \). A finite subset of \( \mathcal{C}_A \) is consistent if its down-closure is consistent in \( A \parallel A \).

The characterisation of configurations of \( \mathcal{C}_A \) reveals an important partial order
on configurations of $A$. Let $x$ and $y$ be configurations of an event structure with polarity. Write $x \preceq^- y$ to mean $x \subseteq y$ and $\mathit{pol}(y \setminus x) \subseteq \{-\}$, i.e. the configuration $y$ extends the configuration $x$ solely by events of $-$ve polarity. Similarly, $x \preceq^+ y$ means $x \subseteq y$ and $\mathit{pol}(y \setminus x) \subseteq \{+\}$. Use $\preceq^-$ to denote the converse order to $\preceq^-$. Define the \textit{Scott order} on configurations by $x \preceq y$ iff $x \preceq^- \cdot + \preceq^- \cdots \preceq^- y$. Then, $\preceq$ is a partial order and part of a factorization system: if $x \preceq y$ then $\exists ! z. x \preceq^- z \preceq^+ y$.

\textbf{Proposition 3.1} \cite{19} Let $A$ be a game. Then, $x \in \mathcal{C}(\mathbb{C} A)$ iff $x_2 \subseteq_A x_1$, where $x_1 \in \mathcal{C}(A^+) \text{ and } x_2 \in \mathcal{C}(A)$ are the projections of $x \in \mathcal{C}(A^+ \parallel A)$ to its components.

\textbf{Theorem 3.2} \cite{19} Strategies $\sigma : S \rightarrow A$ correspond to discrete fibrations denoted $\sigma^\ast : (\mathcal{C}(S), \preceq_S) \rightarrow (\mathcal{C}(A), \preceq_A)$, preserving $\preceq^-$, $\preceq^+$ and $\emptyset$.

The theorem says we can view strategies in a game as (certain) discrete fibrations, so equivalently as presheaves over finite configurations with the Scott order. In particular, a strategy from a game $A$ to a game $B$ corresponds to a presheaf over $(\mathcal{C}(A^+ \parallel B), \preceq_A \parallel \preceq_B) \cong (\mathcal{C}(A), \preceq_A)^{\text{op}} \times (\mathcal{C}(B), \preceq_B)$, so to a profunctor from $(\mathcal{C}(A), \preceq_A)$ to $(\mathcal{C}(B), \preceq_B)$. This correspondence yields a lax functor from strategies to profunctors. The view of strategies as (special) profunctors — explained further in \cite{19} — will guide our later work.

We obtain a bicategory of concurrent games and strategies in which the objects are event structures with polarity — the games, the arrows from $A$ to $B$ are strategies $\sigma : A \rightarrow B$ and the 2-cells are maps of strategies. The vertical composition of 2-cells is the usual composition of maps. Horizontal composition is given by the composition of strategies $\circ$ (which extends to a functor on 2-cells via the universality of pullback).

A strategy in the game $A^+ \parallel B$ corresponds to a strategy in the game $(B^+)^+ \parallel A^+$. Hence a strategy $A \rightarrow B$ corresponds to a dual strategy $B \rightarrow A^+$. The bicategory is rich in structure, in particular, it is compact-closed (so has a trace, a feedback operation).

\section{3.2 Operations on strategies and duplication}

Beyond composition there are many other useful operations on strategies. Several of these have appeared previously in, for example, establishing determinacy \cite{3} or the value theorem for games with pay-off \cite{4} where proofs often hinge on constructing appropriate strategies.

We can form the \textit{sum of strategies} $\bigsqcup_{i \in I} \sigma_i$ of a family of strategies $\sigma_i : S_i \rightarrow A$, $i \in I$, in a common game $A$ \cite{4}. This is formed as the sum of the event structures $S_i$ but where the initial $-$ve events are identified to maintain receptivity. A sum of strategies only commits to a particular component strategy once a Player move is made there. The empty sum $\bot$ essentially comprises the initial segment of the game $A$ consisting of all the initial $-$ve events of $A$.

The pullback of a strategy $\sigma$ across a (possibly partial) map $f$ of event structures is itself a strategy $f^\ast \sigma$ \cite{18}:

\[
\begin{array}{ccc}
S' & \xrightarrow{f^\ast \sigma} & S \\
\downarrow & & \downarrow \sigma \\
A & \xrightarrow{f} & B.
\end{array}
\]
This operation can adjoin extra events and causal links to the original strategy; it
subsumes, for example, operations on strategies in which we prefix an initial event.

We shall also use a previously unnoticed strategy $\delta_A$ that exists from a game
$A \rightarrow A \parallel A$, and expresses a form of duplication. Like copy-cat it introduces extra
‘causal wiring’ but its definition is much more subtle, though is easy to describe
in special cases. For example if the game $A$ comprises a single Player move $\oplus$ the
strategy $\delta_A : A \rightarrow A \parallel A$ takes the form $\oplus \leftarrow \oplus$. A similar construction applies in the
case where all the moves of a game are that of Player. If the game $A$ comprises
a single Opponent move $\ominus$ then $\delta_A : A \rightarrow A \parallel A$ takes the form $\ominus \leftarrow \ominus$, where the
wiggly line indicates the inconsistency between the two Player moves.

**Duplication** The definition of $\delta_A : A \rightarrow A \parallel A$ in general is via rigid families. For
each triple $(x, y_1, y_2)$, where $x \in C(A^i)$ and $y_1, y_2 \in C(A)$, which is balanced, i.e.
$\forall a \in y_1 \cup y_2. \text{pol}_A(a) = + \implies a \in x$ and $\forall a \in x. \text{pol}_{A^i}(a) = + \implies a \in y_1$ or $a \in y_2$,
and choice function $\chi : x^+ \rightarrow \{1, 2\}$, from the positive events of $x$ denoted by $x^+$,
such that $\chi(a) = 1 \implies a \in y_1$ and $\chi(a) = 2 \implies a \in y_2$, the order $q(x, y_1, y_2; \chi)$ is
defined to have underlying set $\{(0, a), (1, a) \mid a \in y_1\} \cup \{(0, a), (2, a) \mid a \in y_2\} \cup$
$\{((\chi(a), a), (0, a)) \mid a \in x \& \text{pol}_{A^i}(a) = +\}$.

The rigid family $Q$ consists of all such $q(x, y_1, y_2; \chi)$ for balanced $(x, y_1, y_2)$ and
choice functions $\chi$. From $Q$ we obtain the event structure $\text{Pr}(Q)$ in which events
are prime orders, i.e. with a top element; events of $\text{Pr}(Q)$ inherit the polarity
of their top elements to obtain an event structure with polarity. We define the
strategy $\delta_A : A \rightarrow A \parallel A$ to be the map $\text{Pr}(Q) \rightarrow A^i \parallel A \parallel A$ sending a prime to its top
element. We remark that the meaning of a triple of configurations $x, y_1, y_2$ of $C$
being balanced is almost $y_1 \cup y_2 \subseteq C \ x$ but is not this in general as $y_1 \cup y_2$ need not
itself be a configuration of $C$.

The operation $\delta_A$ forms a comonoid with counit $\bot : A \rightarrow \emptyset$. In fact, they form part
of a Frobenius algebra in which the accompanying monoid is obtained by duality.

### 4 A language for strategies

We describe, somewhat schematically, a language for describing strategies based on
the constructions above. In fact, it is based on an earlier language for profunctors [15],
taking advantage of the view of strategies as special profunctors.

#### 4.1 Types

Types are games $A, B, C, \cdots$. We have type operations corresponding to the opera-
tions on games of forming the dual $A^i$, simple parallel composition $A \parallel B$, sum $\Sigma_{i \in I} A_i$
as well as recursively-defined types $\mu X. A(X)$, although we shall largely ignore the
latter as it rests on well-established techniques [14].

One way to relate types is through the affine maps between them. There will be operations for shifting between types related by affine maps (described by configuration expressions), enabling us to shift strategies either forwards or backwards across affine maps.

A **type environment** is a finite partial function from variables to types, for convenience written typically as \( \Gamma \equiv x_1 : A_1, \ldots, x_m : A_m \), in which the (configuration) variables \( x_1, \ldots, x_m \) are distinct. It denotes a (simple) parallel composition \( \|_{x_i : A_i} \). In describing the semantics we shall sometimes write \( \Gamma \) for the parallel composition it denotes.

### 4.2 Configuration expressions

Configuration expressions denote finite configurations of games in an environment. A typing judgement \( \Gamma \vdash p : B \) for a configuration expression \( p \) in a type environment \( \Gamma \) denotes an affine map from \( \Gamma \) to \( B \). In particular, the judgement \( \Gamma, x : A \vdash x : A \) denotes the partial map of event structures projecting to the single component \( A \). The special case \( x : A \vdash x : A \) denotes the identity map.

We shall allow configuration expressions to be built from affine maps \( f = (f_0, f_1) : A \to B \) in the judgement \( \Gamma, x : A \vdash f : B \) and the equivalent judgement \( \Gamma, x : A \vdash f_0 \cup f_1 : B \). In particular, \( f_1 \) may be completely undefined, allowing configuration expressions to be built from constant configurations, as e.g. in the judgement for the empty configuration \( \Gamma \vdash \emptyset : A \) or a singleton configuration \( \Gamma \vdash \{a\} : A \) when \( a \) is an initial event of \( A \). The expression \( \{a\} \cup x' \) associated with the judgement \( \Gamma, x' : A/a \vdash \{a\} \cup x' : A \), where \( a \) is an initial event of \( A \), is used later in the transition semantics.

The three inductive rules for configuration expressions are as follows, where \( \Gamma^k \) is \( x_1 : A_1, \ldots, x_m : A_m :\)

\[
\begin{align*}
\Gamma \vdash p : A_j & \quad j \in I \\
\Gamma \vdash jp : \sum_{i \in I} A_i & \\
\Gamma \vdash p : A & \quad \Delta \vdash q : B & \quad \Gamma \vdash p : B \\
\Gamma, \Delta \vdash (p, q) : A \| B & \\
\Gamma^k \vdash p : B^k & 
\end{align*}
\]

For a sum \( \sum_{i \in I} A_i \), the left-hand rule gives configuration expressions \( jp \) where \( j \in I \) and \( p \) is a configuration expression of type \( A_j \); In the central rule for simple parallel composition we exploit the fact that configurations of simple parallel compositions are essentially pairs of configurations of the components. Finally, in the right-hand rule configurations of \( B^k \) can be taken to be the same as configurations of \( B \).

### 4.3 Terms for strategies

Terms denoting strategies have typing judgements:

\[
x_1 : A_1, \ldots, x_m : A_m \vdash t \rightharpoonup y_1 : B_1, \ldots, y_n : B_n ,
\]

where all the variables are distinct, interpreted as a strategy from the game \( x_1 : A_1, \ldots, x_m : A_m \) denotes to the game \( y_1 : B_1, \ldots, y_n : B_n \) denotes.

We can think of the term \( t \) as a box with input and output wires for the typed variables:
Duality The duality of input and output is caught by the rules:
\[ \Gamma, x : A \vdash t \rightarrow \Delta \quad \Gamma \vdash t \rightarrow x : A, \Delta \]
\[ \Gamma \vdash t \rightarrow x : A^i, \Delta \quad \Gamma, x : A^i \vdash t \rightarrow \Delta \]

Composition The composition of strategies is described in the rule
\[ \Gamma \vdash t \rightarrow \Delta \quad \Delta \vdash u \rightarrow H \]
\[ \Gamma \vdash \exists \Delta, [t \parallel u] \rightarrow H \]

which, in the picture of strategies as boxes, joins the output wires of one strategy to input wires of the other. Note that the simple parallel composition of strategies arises as a special case when \( \Delta \) is empty.

Nondeterministic sum We can form the nondeterministic sum of strategies of the same type:
\[ \Gamma \vdash t_i \rightarrow \Delta \quad i \in I \]
\[ \Gamma \vdash \bigsqcup_{i \in I} t_i \rightarrow \Delta \]

We shall use \( \bot \) for the empty nondeterministic sum, when the rule above specialises to \( \Gamma \vdash \bot \rightarrow \Delta \). The term \( \bot \) denotes the minimum strategy in the game \( \Gamma^i \parallel \Delta \).

Pullback We can form the pullback of two strategies of the same type:
\[ \Gamma \vdash t_1 \rightarrow \Delta \quad \Gamma \vdash t_2 \rightarrow \Delta \]
\[ \Gamma \vdash t_1 \land t_2 \rightarrow \Delta \]

In the case where \( t_1 \) and \( t_2 \) denote the respective strategies \( \sigma_1 : S_1 \rightarrow \Gamma^i \parallel \Delta \) and \( \sigma_1 : S_1 \rightarrow \Gamma^i \parallel \Delta \), the strategy \( t_1 \land t_2 \) denotes the pullback of \( \sigma_1 \) and \( \sigma_2 \). Informally, such a strategy acts as the two component strategies agree to act.

Hom-set The hom-set rule is a powerful way to lift affine maps or relations expressed in terms of cospans of affine maps to strategies. Write \( p[\emptyset] \) for the substitution of the empty configuration \( \emptyset \) for all configuration variables appearing in a configuration expression \( p \). The hom-set rule
\[ \Gamma \vdash p' : C \quad \Delta \vdash p : C \]
\[ p[\emptyset] \in_C p'[\emptyset] \]
introduces a term standing for the hom-set \( (C(C), \in_C)(p, p') \). It relies on configuration expressions \( p, p' \) and their typings. If \( \Delta \vdash p : C \) denotes the affine map \( g = (g_0, g_1) \) and \( \Gamma \vdash p' : C \) denotes the affine map \( f = (f_0, f_1) \), the side condition of the rule ensures that \( g_0 \in_C f_0 \). A term for copy-cat arises as a special case of the hom-set rule:
\[ x : A \vdash y \in_A x \rightarrow y : A \]

The hom-set rule is very expressive — see Section 4.4. The precise definition of the strategy which the hom-set rule yields is given in the next section.

Duplication Duplication terms are described by the rule
\[ \Gamma \vdash p : C \quad \Delta_1 \vdash q_1 : C \quad \Delta_2 \vdash q_2 : C \]
\[ \Gamma \vdash \delta_C(p, q_1, q_2) \rightarrow \Delta_1, \Delta_2 \]
provided \( p[\emptyset], q_1[\emptyset], q_2[\emptyset] \) is balanced in the sense of Section 3.2. The term for the duplication strategy is, in particular, \( x : A \vdash \delta_A(x, y_1, y_2) \rightarrow y_1 : A, y_2 : A \).
4.3.1 Hom-set terms: semantics

The definition of the strategy which \( \Gamma \vdash p \ll C \ p' \vdash \Delta \) denotes is quite involved. We first simplify notation. W.l.o.g. assume \( \Delta \vdash p : C \) and \( \Gamma \vdash p' : C \). Using duality we can always rearrange the environment to achieve this. Write \( A \) for the denotation of the environment \( \Gamma \) and \( B \) for the denotation of \( \Delta \). Let \( \Delta \vdash p : C \) and \( \Gamma \vdash p' : C \) denote respectively the affine maps \( g = (g_0, g_1) : B \to_a C \) and \( f = (f_0, f_1) : A \to_a C \). Note that we have that \( g_0 \ll C \ f_0 \) from the typing of \( p \ll C \ p' \).

We build the strategy out of a rigid family \( Q \) with elements as follows. First, define a pre-element to be a finite preorder comprising a set \( \{1\} \times x \cup \{2\} \times y \), for which \( x \in C(A^1) \) \& \( y \in C(B) \) \& \( gy \ll f \ x \), with order that induced by \( \leq_A \) on \( x \), \( \leq_B \) on \( y \), with additional causal dependencies

\[
(1, a) \leq (2, b) \text{ if } f_1(a) = g_1(b) \text{ & } b \text{ is +ve, and } \\
(2, b) \leq (1, a) \text{ if } f_1(a) = g_1(b) \text{ & } b \text{ is -ve.}
\]

As elements of the rigid family \( Q \) we take those pre-elements for which the order \( \leq \) is a partial order (i.e. is antisymmetric). The elements of \( Q \) are closed under rigid inclusions, so \( Q \) forms a rigid family. We now take \( S = \text{def } \Pr(Q) \); the events of \( S \) (those elements of \( Q \) with a top event) map to their top events in \( A^1 \parallel B \) from where they inherit polarities. This map can be checked to be a strategy: innocence follows directly from the construction, while receptivity follows from the constraint that \( gy \ll f \ x \).

It is quite easy to choose an example where antisymmetry fails in a pre-element, in other words, in which the preorder is not a partial order. However, when either \( p \) or \( p' \) is just a variable, no nontrivial causal loops are introduced and all pre-elements are elements. More generally, if one of \( p \) or \( p' \) is associated with a partial rigid map (i.e. a map which preserves causal dependency when defined), then no nontrivial causal loops are introduced and all pre-elements are elements.

4.3.2 Duplication terms: semantics

Consider now the semantics of a term \( \Gamma \vdash \delta_C(p, q_1, q_2) \vdash \Delta \). W.l.o.g. we may assume that the environment is arranged so \( \Delta = \Delta_1, \Delta_2 \) with judgements \( \Gamma \vdash p : C \), \( \Delta_1 \vdash q_1 : C \) and \( \Delta_2 \vdash q_2 : C \). To simplify notation assume the latter judgements for configuration expressions denote the respective affine maps \( f = (f^0, f^1) : A \to_a C \), \( g_1 = (g_1^0, g_1^1) : B_1 \to C \) and \( g_2 = (g_2^0, g_2^1) : B_2 \to C \). From the typing of \( \delta_C(p, q_1, q_2) \) we have that \( (f^0, g_1^0, g_2^0) \) forms a balanced triple in \( C \). We build the strategy out of a rigid family \( Q \) with elements as follows. We construct pre-elements from \( x \in C(A^1), y_1 \in C(B_1) \) and \( y_2 \in C(B_2) \) where \( (f \ x, g_1 \ y_1, g_2 \ y_2) \) is a balanced triple in \( C \) with a choice function \( \chi \). There are three kinds of elements of \( x \):

\[
x_+ = \{ a \in x \mid \text{pol}_{A^1}(a) = 1 \}, \\
x_0 = \{ a \in x \mid \text{pol}_{A^1}(a) = 0 \} \text{ and } f^1(a) \in g_0^0, \\
x_+ = \{ a \in x \mid \text{pol}_{A^1}(a) = 1 \} \text{ and } f^1(a) \in g_1^0, g_2^0,\}
\]

We define a typical pre-element to be a finite preorder on the set

\[
\{0\} \times (x_+ \cup x_0) \cup \{ (\chi(f^1(a)), a) \mid a \in x_0 \} \cup \{1\} \times y_1 \cup \{2\} \times y_2,
\]

102
with order that induced by that of the game $A^i \parallel B_1 \parallel B_2$ with additional causal dependencies

$$(0,a) \leq (1,b) \text{ if } f^1(a) = g^1_1(b) \& b \text{ is } +\text{ve in } B_1,$$

$$(0,a) \leq (2,b) \text{ if } f^1(a) = g^1_2(b) \& b \text{ is } +\text{ve in } B_2, \text{ and}$$

$$(\chi(f^1(a)), b) \leq (0,a) \text{ if } a \in x^*_1 \& f^1(a) = g^1_\chi(f^1(a))(b),$$

for $b$ $-\text{ve in } B_{\chi(f^1(a))}$. As elements of the rigid family $Q$ we take those pre-elements for which the order $\leq$ is a partial order (i.e. is antisymmetric). Once $Q$ is checked to be a rigid family, we can take $S \defeq \Pr(Q)$; the events of $S$ map to the events in the game $A^i \parallel B_1 \parallel B_2$ associated with their top events, from where they inherit polarities. This map defines the strategy denoting the original duplication term.

### 4.4 Expressivity

The terms for strategies are surprisingly expressive and potentially rich in laws, able to support a form of equational reasoning, that we can only touch on here. For example the Frobenius algebra associated with duplication immediately yields laws. Other laws capture basic facts about the Scott order. For instance, assuming $z \subseteq x, y$ in $C(A)$, we have $y \subseteq_A x$ iff $y/z \subseteq_{A/j} x/z$.

As we shall see, we can derive the laws expected of a recursion operator provided the recursion involves a homomorphism w.r.t. the duplication comonad, and this fact too we could hope to derive. Some of the reasoning can be made diagrammatic, using the techniques of string diagrams.

Hom-set terms provide many basic strategies. The denotation of $x : A \vdash \emptyset \in_A \emptyset \vdash y : B$ is the strategy in the game $A^i \parallel B$ given by the identity map $\text{id}_{A \parallel B} : A^i \parallel B \to A^i \parallel B$. The denotation of $\vdash y \in_A \emptyset \vdash y : A$ is $\bot_A$, the minimum strategy in the game $A$ comprising just the initial $-\text{ve}$ events of $A$.

The judgement $x : A_j \vdash y \in_{\Sigma_{i \in I} A_i} j x \vdash y : \Sigma_{i \in I} A_i$ denotes the injection strategy: its application to a strategy in $A_j$ fills out the strategy according to the demands of receptivity to a strategy in $\Sigma_{i \in I} A_i$. Its converse $x : \Sigma_{i \in I} A_i \vdash j y \in_{\Sigma_{i \in I} A_i} x \vdash y : A_j$ applied to a strategy of $\Sigma_{i \in I} A_i$ projects the strategy to a strategy in $A_j$.

More is obtained by combining hom-set with other operations such as composition. Assume $\vdash t \vdash y : B$. When $f : A \to B$ is a map of event structures with polarity, the composition $\vdash \exists y : B.\lfloor t \parallel fx \in_B y \rfloor \vdash x : A$ denotes the pullback $f^*\sigma$ of the strategy $\sigma$ denoted by $t$ across the map $f : A \to B$. In the case where a map of event structures with polarity $f : A \to B$ is innocent, the composition $\vdash \exists x : A.\lfloor y \in_B fx \parallel t \rfloor \vdash y : B$ denotes the ‘relabelling’ $f_!\sigma$ of the strategy $\sigma$ denoted by $t$.

A great deal is achieved through basic manipulation of the input and output “wiring” afforded by the hom-set rules and input-output duality. For instance, to achieve the effect of lambda abstraction: via the hom-set rule we obtain

$$x : A^i, y : B \vdash z \in_{A \parallel B} (x, y) \vdash z : A^i \parallel B,$$
which joins two inputs to a common output, whence:

\[
\Gamma, x : A \vdash t \rightarrow y : B \quad \vdash : A^1, y : B \vdash z \in A \parallel B (x, y) \rightarrow z : A^1 \parallel B
\]

\[
\Gamma \vdash \exists x : A^1, y : B. [t \parallel z \in A \parallel B (x, y)] \rightarrow z : A^1 \parallel B
\]

A trace, or feedback, operation is another consequence of such “wiring.” Given a strategy \(\Gamma, x : A \vdash t \rightarrow y : A, \Delta\) represented by the diagram

\[
\begin{array}{c}
\Gamma \\
\downarrow t \\
A \\
\uparrow \Delta
\end{array}
\]

we obtain

\(\Gamma, \Delta^1 \vdash t \rightarrow x : A^1, y : A\), which, post-composed with the term \(x : A^1, y : A \vdash x \subseteq A \ y \rightarrow\) denoting the copy-cat strategy \(\gamma_{A^1}\), yields \(\Gamma \vdash \exists x : A^1, y : A. [t \parallel x \subseteq A \ y] \rightarrow \Delta\), representing its trace:

\[
\begin{array}{c}
\Gamma \\
\downarrow t \\
A \\
\uparrow \Delta
\end{array}
\]

The composition introduces causal links from the +ve events of \(y : A\) to the −ve events of \(x : A\), and from the +ve events of \(x : A\) to the −ve events of \(y : A\) — these are the usual links of copy-cat \(\gamma_{A^1}\) as seen from the left of the turnstyle. This trace coincides with the feedback operation which has been used in the semantics of nondeterministic dataflow (where only games comprising solely Player moves are needed) [13].

Recursive definitions can be achieved from trace with the help of duplication. For those strategies which respect \(\delta\), i.e. \(\delta_A \circ \sigma \cong (\sigma \parallel \sigma) \circ \delta_{\Gamma \parallel A}\), and in particular for strategies which are homomorphisms between \(\delta\)-comonoids, the recursive definition does unfold in the way expected, in the sense that the recursive definition is isomorphic to its unfolding:

\[
\begin{array}{c}
\Gamma \sigma \\
\delta_A
\end{array} \cong \begin{array}{c}
\Gamma \delta_{\Gamma} \\
\sigma \\
\delta_A
\end{array} \sigma \\

A
\]

This follows as a general fact from the properties of trace monoidal categories and the string-diagram reasoning they support. However, not all strategies are homomorphisms between \(\delta\)-comonoids, characterised in the following theorem.

**Theorem 4.1** A strategy \(\sigma : S \rightarrow A^1 \parallel B\) respects \(\delta\) iff

- components \(\sigma_1 : S \rightarrow A^1\) and \(\sigma_2 : S \rightarrow B\) preserve causal dependency when defined,
- \(\sigma_1\) reflects configurations of \(A^1\), i.e. if \(x \subseteq S\) is such that \(\sigma_1 x \in C(A^1)\) then \(x \in C(S)\), and
- for every +ve event \(s \in S\) such that \(\sigma(s) \in A^1\) we have the number of −ve events of \(\sigma_2[s]\) equals the number of +ve events of \(\sigma_1[s]\).

In this case, \(\sigma\) also respects counits, i.e. \(\bot_B \circ \sigma \cong \bot_A\).
5 A process perspective

The operations on strategies have much in common with the operations of process algebra and can be seen as forming the basis of a higher-order process language. However, from the perspective of concurrent processes, we must address several issues: its operational semantics, a suitable form of equivalence and expressivity. These require we examine the effects of synchronization and the internal, neutral events it produces, more carefully. Composition of strategies can introduce deadlock which is presently hidden. This, for example, affects the reliability of winning strategies; presently a strategy may be deemed winning and yet possibly deadlock before reaching a winning configuration. The next example illustrates how hidden deadlocks may be created in a composition of strategies.

Example 5.1 (i) Deadlock may arise in a composition \( \tau \circ \sigma \) through \( \sigma : A \rightarrow C \) and \( \tau : B \rightarrow C \) imposing incompatible causal dependencies between events in \( B \). For instance \( B \) may contain two concurrent events of opposite polarities \( b_1 = \ominus \) and \( b_2 = \oplus \). The strategy \( \sigma \) may impose the causal dependency \( s_1 \rightarrow s_2 \) between occurrences of \( b_1 \) and \( b_2 \) respectively. From the point of view of strategy \( \tau \), the game \( B \) has changed polarity to \( B \perp \) and \( \tau \) may impose the reverse causal dependency \( s_2 \rightarrow s_1 \) between occurrences of \( b_2 \) and \( b_1 \) respectively.

(ii) Composition of strategies may hide computation which is stuck. For games \( B = \ominus \parallel \ominus \) and \( C = \ominus \), assume strategy \( \sigma_1 : \ominus \rightarrow B \) nondeterministically chooses the right or left move in \( B \), strategy \( \sigma_2 : \ominus \rightarrow B \) chooses just the right move in \( B \), while strategy \( \tau : B \rightarrow C \) yields output in \( C \) if it gets the right event of \( B \) as input. The two strategy compositions \( \tau \circ \sigma_1 \) and \( \tau \circ \sigma_2 \) are indistinguishable. 

5.1 Partial strategies

To treat such phenomena explicitly and in order to obtain a transition semantics, we extend strategies with neutral events. Extend event structures with polarity to include a neutral polarity \( 0 \); as before, maps preserve polarities when defined. However within games we shall still assume that all events have +ve or −ve polarity.

Definition 5.2 A partial strategy from a game \( A \) to a game \( B \) comprises a total map \( \sigma : S \rightarrow A \perp \parallel B \) of event structures with polarity (in which \( S \) may also have neutral events) where

(i) \( N \) is an event structure consisting solely of neutral events;
(ii) \( \sigma \) is receptive, i.e. if \( \sigma x \xrightarrow{c} \) in \( C(S) \) with \( c \) −ve, then \( x \xrightarrow{s} \) and \( \sigma(s) = c \), for some unique \( s \in S \);
(iii) in the partial-total factorization of the composition of \( \sigma \) with the projection \( A \perp \parallel N \parallel B \rightarrow A \perp \parallel B \), drawn below, the defined part \( \sigma_0 \) is a strategy.

\[
\begin{array}{ccc}
S & \xrightarrow{\sigma} & S_0 \\
\downarrow & & \downarrow \sigma_0 \\
A \perp \parallel N \parallel B & \rightarrow & A \perp \parallel B
\end{array}
\]

Partial strategies in a game \( A \) correspond to partial strategies from the empty game to \( A \). Strategies between games correspond to those partial strategies in which
the neutral events \( N \) form the empty event structure.

It may seem odd that partial strategies are total maps. Why have we not taken a partial strategy to be undefined on events which are sent to \( N \)? Because such partial maps do not behave well under pullback, and this would complicate the definition of composition and spoil later results such as that the pullback of a partial strategy is a partial strategy. With our choice of definition we are able to localise neutral events to the games over which they occur; with the alternative definition, different forms of undefined would become conflated.

5.2 Operations on partial strategies

The operations on strategies extend and give an interpretation of the language of Section 4 in terms of partial strategies. The defined parts of the operations on partial strategies coincide with the operations on the defined parts.

We can compose two partial strategies \( \sigma : S \to A^1 \parallel N_S \parallel B \) and \( \tau : T \to B^1 \parallel N_T \parallel C \) by pullback. Ignoring polarities temporarily, and padding with identity maps, we obtain \( \tau / \text{uni} \sigma \) via the pullback

\[
\begin{array}{ccc}
S \parallel N_T \parallel C & \xrightarrow{\tau \text{ / uni} \sigma} & A \parallel N_S \parallel T \\
\sigma \parallel N_T \parallel C & \xrightarrow{\text{uni} \sigma} & A \parallel N_S \parallel B \parallel N_T \parallel C.
\end{array}
\]

once we reinstate polarities and make the events of \( B \) neutral. Receptivity of \( \tau \text{ / uni} \sigma \) follows directly from that of \( \sigma \) and \( \tau \). That the defined part of \( \tau \text{ / uni} \sigma \) is a strategy follows once it is shown that the defined part of the composite \( T \oplus S \xrightarrow{\tau \oplus \sigma} A^1 \parallel (N_S \parallel B \parallel N_T) \parallel C \to A^1 \parallel C \) is isomorphic to \( \tau_0 \oplus \sigma_0 \), the composition of the defined parts of \( \sigma \) and \( \tau \).

With partial strategies we no longer generally have that composition with copy-cat yields the same strategy up to isomorphism: there will generally be extra neutral events introduced through synchronizations. However, as demonstrated in Section 6, a bicategory may be recovered through the use of may/must equivalence.

Let \( \sigma_i : S_i \to A^1 \parallel N_i \parallel B \), where \( i \in I \), be a family of partial strategies. Their sum is the partial strategy \( \bigsqcup_{i \in I} \sigma_i : S \to A^1 \parallel (\bigsqcup_{i \in I} N_i) \parallel B \). Its events are obtained as the disjoint union of the \( S_i \) but where the initial -ve events are identified to maintain receptivity; they map under \( \bigsqcup_{i \in I} \sigma_i \) as directed by the component maps \( \sigma_i \). Causal dependency is inherited from the components \( S_i \) with a finite subset of events consistent iff its down-closure contains +ve events from at most one \( S_i \). As such, the nondeterministic sum only commits to a component through the occurrence of a positive event: from the perspective of tracking potential deadlocks, it is not necessary to view neutral events as committing to a particular component since, in isolation, a neutral event cannot introduce deadlock when composed with a counterstrategy due to receptivity.
The pullback of partial strategies \( \sigma_1 : S_1 \to A^1\|N_1\|B \) and \( \sigma_2 : S_2 \to A^1\|N_2\|B \)
is obtained as to the right.

\[
\begin{array}{c}
S_1 \wedge S_2 \\
\downarrow \sigma_1 \wedge \sigma_2 \\
S_1\|N_2 \\
\downarrow \sigma_1 \| N_2 \\
A^1\|N_1\|B \| p_2 \| N_1
\end{array}
\]

5.3 Transition semantics

We now discuss transition semantics for partial strategies presented in Figure 1. For brevity, the rules presented require the left-hand environment to be empty; this can always be achieved using the rules for duality. In the transition rules, we write \( t \vdash \Delta \) instead of \( \varnothing \vdash t \vdash \Delta \). Transitions are associated with two kinds of actions, either an action \( o \) associated with a hidden neutral action \( t \vdash \Delta \xrightarrow{\sigma} t' \vdash \Delta \) or an initial event located in the environment \( t \vdash x : A, \Delta \xrightarrow{x:a;x'} t' \vdash x' : A/a, \Delta \). Notice that a neutral action leaves the types unchanged but may affect the term. An action \( x : a : x' \) is associated with an initial event \( ev(x : a : x') = \text{def} x : a \) at the \( x \)-component of the environment. On its occurrence, the component of the environment \( x : A \) is updated to \( x' : A/a \) in which \( x' \), a fresh resumption variable, stands for the configuration remaining in the remaining game \( A/a \). Say an action \( x : a : x' \) is +ve/−ve according as \( a \) is +ve/−ve. In the rules for composition, we use \( \alpha \) for \( o \) or an action of the form \( x : a : x' \) where \( x \) is in the domain of \( \Gamma \) and use \( \beta \) for \( o \) or an action of the form \( y : b : y' \) where \( y \) is in the domain of \( H \).

In typed judgements of \( \delta_C(p, q_1, q_2) \), a variable can appear free in at most one of \( p, q_1, q_2 \). Write, for example, \( y \in \text{fv}(p) \) for \( y \) is free in \( p \), and \( q_1(y : b) \in p[\varnothing] \) to mean the image of \( b \) under the map \( q_1 \) denotes is in the configuration denoted by \( p[\varnothing] \).

Theorem 5.3 Assume certain primitive strategies \( \varnothing \vdash \sigma_0 \vdash \Delta \), so as a map, \( \sigma_0 : S \to \Delta \), for which we assume rules,

\[
\sigma_0 \vdash \Delta \xrightarrow{\epsilon} \sigma_0' \vdash \Delta'
\]

is initial in \( S \), \( \sigma_0(s) = ev(\epsilon) \).

Then, derivations in the operational semantics

\[
t \vdash \Delta \xrightarrow{\epsilon} t' \vdash \Delta',
\]

up to \( \alpha \)-equivalence, in which \( t \) denotes the partial strategy \( \sigma : S \to \Delta \), are in 1-1 correspondence with initial events \( s \) in \( S \) such that \( \sigma(s) = ev(\epsilon) \) when \( ev(\epsilon) \neq o \) or \( s \) is neutral when \( ev(\epsilon) = o \).

6 May and Must equivalence

We now study ‘may’ and ‘must’ equivalence and how it may be used to recover bicategorical structure on strategies equipped with stopping configurations.

Consider three partial strategies in a game comprising a single +ve event. For this discussion it will be sufficient to consider just the event structures of the partial strategies:

\[
S_1 \circ S_2 \circ \rightarrow S_3 \circ \rightarrow
\]
Composition:
\[
\begin{align*}
t \to y : B, \Delta, \Gamma & \xrightarrow{y \in y'} t' \to y' : B/h, \Delta, \Gamma \\
u \to y : B', \Delta', H & \xrightarrow{y \in y'} u' \to y' : B'/h, \Delta', H
\end{align*}
\]
\[
\exists y : B, \Delta, \Gamma \parallel u \to y' : B/h, \Delta, \Gamma
\]

Hom-sets: Assuming \( a \) is an initial event of \( A \) for which \( p((a) \parallel [\emptyset]) \subseteq p'((a) \parallel [\emptyset]) \).
\[
p \subseteq C p' \to x : A, \Delta \xrightarrow{x, a \in x'} p((a) \cup x' / x) \subseteq C p'((a) \cup x' / x) \to x' : A / a, \Delta
\]
Above, the variable \( x \) will only appear in one of \( p \) and \( p' \).

**Sum of partial strategies:**
\[
\begin{align*}
t_i \to \Delta \xrightarrow{\epsilon} t_i' + \Delta' & \forall i \in I, \epsilon \text{ is } -\text{ve} \\
\prod_{i \in I} t_i \to \Delta \xrightarrow{\epsilon} \prod_{i \in I} t_i' + \Delta
\end{align*}
\]

**Pullback:**
\[
\begin{align*}
t_1 \to \Delta \xrightarrow{\alpha} t_1' \to \Delta & \\
t_2 \to \Delta \xrightarrow{\alpha} t_2' \to \Delta & \\
t_1 \wedge t_2 \to \Delta \xrightarrow{\alpha} t_1' \wedge t_2' \to \Delta
\end{align*}
\]

**Duplication:**
\[
\delta_C(p, q_1, q_2) \to x : A, \Delta \xrightarrow{x, a \in x'} \delta_C(p, q_1, q_2)((a) \cup x' / x) \to x' : A / a, \Delta \text{ if either}
\]
- \( a \) is an initial \(-\text{ve} \) event of \( A \), or
- \( a \) is an initial \(+\text{ve} \) event of \( A \) and there exists \( i \in \{1, 2\} \) s.t. either \( x \in \text{fv}(q_i) \) and \( q_i(x / a) \in p([\emptyset]) \) or \( x \in \text{fv}(p) \) and \( p(x / a) \in q_i([\emptyset]) \).

![Fig. 1. Transition semantics](image-url)

Neutral events are drawn as \( \emptyset \). Conflict between pairs of events (meaning that there is no set in the consistency relation containing them both) is drawn as \( \nLeftarrow \).

All three partial strategies have the same strategy as their defined parts. However, from the point of view of observing the move in the game, the first two partial strategies differ from the third. In a maximal play both \( S_1 \) and \( S_2 \) will result in the observation of the single move of the game. However, in \( S_3 \) one maximal play is that in which the leftmost neutral event has occurred, in conflict with observing the single move of the game.

We follow [5] in making these ideas precise. For configurations \( x, y \) of an event structure with polarity which may have neutral events write \( x \notin y \) to mean \( x \subseteq y \) and all events of \( y \setminus x \) have polarity + or 0. We write \( \subseteq_0 \) to mean the inclusion involves only neutral events.

**Definition 6.1** Let \( \sigma \) be a partial strategy in a game \( A \). Let \( \tau : T \to A^1 \parallel \emptyset \) be a \('test'\) strategy from \( A \) to a the game consisting of a single Player move \( \checkmark \).

Say \( \sigma \) may pass \( \tau \) iff there exists \( x \in C^\infty(T \parallel \emptyset) \) with image \((\tau \parallel \sigma)x \) containing \( \checkmark \).

Say \( \sigma \) must pass \( \tau \) iff all \( x \in C^\infty(T \parallel \emptyset) \) which are \( \subseteq_0 \)-maximal have image \((\tau \parallel \sigma)x \) containing \( \checkmark \).

Say two partial strategies are \('may' ('must') equivalent \) iff the tests they may (respectively, must) pass are the same.
Two partial strategies with the same strategy as their defined part are ‘may’
equivalent but need not be ‘must’ equivalent. ‘Must’ inequivalence is lost in moving
from partial strategies to strategies. Moreover, as we have seen, partial strategies
lack identities w.r.t. composition, so they do not even form a bicategory. Fortunately,
for ‘may’ and ‘must’ equivalence it is not necessary to use partial strategies;
it is sufficient to carry with a strategy the extra structure of ‘stopping’ configura-
tions (= images of \(p\)-maximal configurations in a partial strategy). Composition
and copy-cat on strategies extend to composition and copy-cat on strategies with
stopping configurations, while maintaining a bicategory, in the following way.

First, to deal with races, we are forced to introduce a refinement of the Scott
order. We write \(x \triangleleft y\) for the transitive closure of the relation \(\triangleleft \subseteq C^\infty(S) \times C^\infty(S)\)
defined as

\[
x \triangleleft y \iff x \subseteq y \text{ and } y \parallel x \text{ is } +\text{-maximal in } G_S.
\]

On race-free games, \(\triangleleft\) is the identity relation on \(C^\infty(S)\).

Let \(\sigma : S \to A^\perp \| N \| B\) be a partial strategy from a game \(A\) to a game \(B\). Recall
its associated partial-total factorization has defined part a strategy \(\sigma_0 : S_0 \to A^\perp \| B\).
Define the (possibly) stopping configurations in \(C^\infty(S_0)\) to be

\[
\text{Stop}(\sigma) = \defeq \{ d \ x \mid x \in C^\infty(S) \text{ is } p\text{-maximal} \}
\]

where \(d : S \to S_0\) is the partial map that is undefined where \(\sigma\) is undefined and \(\downarrow S\)
is the down-closure of \(S\) for the order \(\triangleleft^*\). Note that \(\text{Stop}(\sigma)\) will include all the
\(+\)-maximal configurations of \(S_0\): any \(+\)-maximal configuration \(y\) of \(S_0\) is the image
under \(p\) of its down-closure \([y]\) in \(S\), and by Zorn’s lemma this extends (necessarily
by neutral events) to a maximal configuration \(x\) of \(S\) with image \(y\) under \(d\); by
maximality, if \(x \rightharpoonup y\) then \(s\) cannot be neutral, nor can it be \(+\)ve as this would
violate the \(+\)-maximality of \(y\).

The operation \(\text{St} : \sigma \mapsto (\sigma_0, \text{Stop}(\sigma))\) above, from partial strategies to strategies
with stopping configurations, motivates the following definitions.

A strategy with stopping configurations in a game \(A\) comprises a strategy \(S \to A\) together with a subset \(M_S \subseteq C^\infty(S)\) which is the down-closure with respect to \(\triangleleft^*\) of a set of \(+\)-maximal configurations. As usual, a strategy with stopping configurations
from a game \(A\) to game \(B\) is a strategy with stopping configurations in
the game \(A^\perp \| B\). We can define ‘may’ and ‘must’ testing of strategies with
stopping configurations analogously to above.

Given two strategies with stopping configurations \(\sigma : S \to A^\perp \| B\), \(M_S\) and \(\tau : T \to B^\perp \| C\), \(M_T\) we define their composition by \((\tau, M_T) \odot (\sigma, M_S) = \defeq (\tau \odot \sigma, M_T \odot M_S)\)
where \(x \in M_T \odot M_S\) iff

\[
\exists z \in C^\infty(T \odot S). \ [x]_{T \odot S} \leq^0 z \text{ and } \Pi_1 z \in M_S \text{ and } \Pi_2 z \in M_T.
\]

The stopping configurations of copy-cat are obtained as for any other strategy, and
in particular \(M_{\text{CC}A} = \{ y \| x \mid x, y \in C^\infty(A) \text{ and } x \triangleleft^* y\}\).

**Proposition 6.2** \(\gamma_A, M_{\text{CC}A}\) is an identity w.r.t. the composition on strategies with
stopping configurations.

**Proposition 6.3** Let \(\sigma\) be a partial strategy from \(A\) to \(B\) and \(\tau\) a partial strategy
from $B$ to $C$. Then

$$\text{Stop}(\tau \circ \sigma) = \text{Stop}(\tau) \circ \text{Stop}(\sigma).$$

**Corollary 6.4** A partial strategy $\sigma$ ‘may’ (respectively ‘must’) pass a test $\tau$ iff $\text{St}(\sigma)$ ‘may’ (‘must’) pass $\tau$. The operation $\text{St}$ preserves ‘may’ and ‘must’ equivalence.

**Example 6.5** It is tempting to think of neutral events as behaving like the internal “tau” events of CCS [10]. However, in the context of strategies they behave rather differently. Consider three partial strategies, over a game comprising of just two concurrent +ve events, say $a$ and $b$. The partial strategies have the following event structures in which we have named events by the moves they correspond to in the game:

$$S_1 \quad a \quad S_2 \quad a \quad S_3 \quad a$$

All three become isomorphic under $\text{St}$ so are ‘may’ and ‘must’ equivalent to each other.

Strategies with stopping configurations inherit the structure of a bicategory from strategies. We can interpret the metalanguage directly in terms of strategies with stopping configurations in such a way that the denotation of a term as a strategy with stopping configurations is the image under $\text{St}$ of its denotation as a partial strategy. To achieve this, we specify the stopping configurations of both the sum and pullback of strategies.

For the sum of strategies $\biguplus_{i \in I} \sigma_i$ with stopping configurations $M_i$, a configuration of the sum is stopping iff it is the image of a stopping configuration under the injection from a component.

Consider strategies $\sigma : S \to A$ and $\tau : T \to A$ with stopping configurations $M_S$ and $M_T$ respectively. Let their pullback be denoted by $\sigma \wedge \tau : P \to A$ with projection morphisms $\pi_1 : P \to S$ and $\pi_2 : P \to T$. A configuration of $P$ is defined to be stopping iff there exist $x_1, x_2$ such that $\pi_1 x \subseteq x_1$ and $\pi_2 x \subseteq x_2$ and $x_1 \in M_S$ and $x_2 \in M_T$, and furthermore there exists a partition $x^+ = Y_1 \cup Y_2$ satisfying $x_i \cap Y_i = \emptyset$. The set of stopping configurations of $P$ coincides with the stopping configurations obtained via $\text{St}$ from the pullback of partial strategies.

The treatment of winning strategies of [3] generalises straightforwardly, with the role of +-maximal configurations replaced by that of stopping configurations.

## 7 Extensions & concluding remarks

We have seen a range of constructions on concurrent strategies that support a rich higher-order language for them, with a corresponding operational semantics. For the latter a central part has been the introduction of partial strategies. Though composition of a partial strategy with copy-cat does not in general yield the same strategy, we have seen how a bicategory can be obtained that respects the may/must behaviour of partial strategies by using stopping configurations.
The bicategorical structure of strategies is largely undisturbed by extensions to probabilistic and quantum games [18], imperfect information [17] and symmetry [1] — though compact-closure becomes *-autonomy under extensions by winning conditions [3] and pay-off [4]. The language of strategies is applicable in these extensions, with minor modifications. The constraints of linearity can be alleviated in games with symmetry which support (co)monads for copying [1]. The constructions of the language extend fairly directly to games with symmetry, though to exploit symmetry fully the language needs to be extended to accommodate (co)monads up to symmetry.

References

On a Categorical Framework for Coalgebraic Modal Logic

Liang-Ting Chen$^{1,2}$

Institute of Information Science  
Academia Sinica  
Taipei, Taiwan

Achim Jung$^3$

School of Computer Science  
University of Birmingham  
Birmingham, U.K.

Abstract

A category of one-step semantics is introduced to unify different approaches to coalgebraic logic parametric in a contravariant functor that assigns to the state space its collection of predicates with propositional connectives. Modular constructions of coalgebraic logic are identified as colimits, limits, and tensor products, extending known results for predicate liftings. Generalised predicate liftings as modalities are introduced. Under common assumptions, the logic of all predicate liftings together with a complete axiomatisation exists for any type of coalgebras, and it is one-step expressive for finitary functors. Colimits and compositions of one-step expressive coalgebraic logics are shown to remain one-step expressive.

Keywords: Predicate liftings, coalgebras, coalgebraic logic, modal logic, Stone duality, compositionality, expressivity

1 Introduction

Two syntax-oriented approaches to coalgebraic modal logic — Moss’ cover modality [23] and Pattinson’s predicate liftings [24,25,26] — are successful in producing a wide range of modal logics parametric in a $\text{Set}$ functor. Subsequently, the algebraic semantics of the logics of predicate liftings was formulated elegantly as a particular form of natural transformations using Stone duality [15]. To explain it, let $\text{BA}$ denote the category of Boolean algebras and $\mathcal{Q}: \text{Set} \to \text{BA}$ the contravariant powerset functor...
functor. For an endofunctor $T$, a family of sets $\Lambda_n$ of $n$-ary predicate liftings indexed by $\mathbb{N}$ amounts to a natural transformation $\delta$ to $UQT$ from the coproduct indexed by all $n$-ary predicate liftings $\lambda$ of $n$-fold product of $UQ$, i.e. $\prod_{n \in \mathbb{N}} \prod_{\lambda \in \Lambda_n} UQ^n$ where $U$ is the forgetful functor, so by adjunction this family $(\Lambda_n)$ gives rise to an interpretation of modalities $\delta: LQ \Rightarrow QT$ for some functor $L$; it introduced what we call one-step semantics (Definition 3.1), an expression first coined by Cîrstea and Pattinson [7] in a different but equivalent form. Later Moss’ cover modality was also formulated in this way [21].

This abstract functorial framework was further developed in [17,4,14,19,18] with the aim of finding suitable modal logics for various coalgebras, e.g. [5,11,13,20], replacing $Q$ with other contravariant functors. For example, Markov processes are coalgebras of the Giry monad, and propositional connectives for measurable spaces can be specified by the contravariant functor $S: \text{Meas} \rightarrow \text{MSL}$ mapping a space to its $\sigma$-algebra, considered as a meet semilattice.

Adequacy and soundness of the functorial framework follow from its very formulation as shown by Kurz [17], and a sufficient condition of expressiveness was first established by Klin [14] for finitary type functors on locally finitely presentable categories (to be presented in Subsection 5.2 below). For example, Boolean logic extended with the possibility modality is expressive for all image-finite Kripke frames, i.e. coalgebras for the finite powerset functor $P_\omega$. The restriction to finitary functors is not necessary, however. Multi-modal logic is expressive for image-finite $A$-labelled Kripke frames, but Klin’s condition does not cover this case, since its corresponding type functor, the $A$-fold product $P_A$, is not necessarily finitary.

Another important line of research investigated modularity of predicate liftings [7,27]: not only expressiveness but also completeness are stable under certain constructions. With regards to the example above, expressiveness of multi-modal logic for $P_A$-coalgebras is inherited from modal logic for $P_\omega$-coalgebras. The idea has since been incorporated into the functorial framework over Stone duality in [19], where a subset of constructions is considered, focusing on completeness. In [18], Kurz and Leal show how to translate Moss’ cover modality into predicate liftings and vice versa; thus making it amenable to their completeness analysis.

In the present paper, we put forward a fully categorical treatment in a syntax-independent fashion beyond Stone duality, so that existing results and concepts can be further applied without further effort to richer structures such as ordered, topological, or measurable spaces. Our running examples will be mostly over $\text{Set}$, however, in the hope that the reader will be able to make a direct comparison with known results and concepts.

In Section 3, we introduce a category $\text{CoLog}$ of one-step semantics, which includes Pattinson’s predicate liftings and Moss’ cover modalities as objects, and exhibit its rich structure. Modularity of coalgebraic modal logics are recognised as standard categorical constructions in this category, avoiding any syntactic bookkeeping. A “full logic” for each type of coalgebras, using the basic properties of adjunction, is identified: every other logic for the same type can be (uniquely) translated to it. In Section 4, we then focus on equational one-step semantics, which are found
to be isomorphic to those determined by predicate liftings. Notions of modalities and equations are derived naturally from the analysis of categorical structures, as is the characterisation of the full equational logic. Logical properties of one-step semantics are discussed in Section 5. Klin’s condition of expressiveness is adapted. The categorical viewpoint allows us to formulate general preservation principles for coalgebraic expressiveness that apply to all logics in CoLog.

The framework is parametric in a contravariant functor mapping state spaces to “algebras” for the base logic about which very little needs to be assumed to cover a large variety of examples. Indeed, we would like to suggest that our work provides the right level of abstraction for understanding coalgebraic modal logic.

This paper summaries the first author’s PhD thesis [6], to which we point for most of the proofs.

2 Preliminaries

We follow Mac Lane’s terminology of category theory [22] except that for us a (co)reflective subcategory is defined to be full. For an endofunctor $T$, a coalgebra for $T$ is a morphism $\xi: X \to TX$; a coalgebra homomorphism from $(X, \xi)$ to $(Y, \gamma)$ is a morphism $f: X \to Y$ satisfying $Tf \circ \xi = \gamma \circ f$. The category of $T$-coalgebras is denoted by $X^T$.

2.1 Dual adjunctions

**Definition 2.1** A contravariant functor $P: \mathcal{X} \to \mathcal{A}$ is said to be dual to a contravariant functor $S$ if together they form an adjunction $S \dashv P: \mathcal{X}^{\text{op}} \to \mathcal{A}$ with unit $\eta$ and counit $\epsilon$.

We use dual adjunctions to set up a link between “state spaces”, the objects of $\mathcal{X}$, and (algebras of) “logics”, the objects of $\mathcal{A}$. This is a particular instance of “Stone duality”; for a general introduction see [12].

**Example 2.2** (i) Consider the powerset $2^{-}$ as a contravariant functor from Set to Set; it is dual to itself. Alternatively, consider the powerset as a Boolean algebra; we obtain $Q$, a contravariant functor from Set to $BA$. The natural dual to $Q$ is the ultrafilter functor $F$, equivalently described as $BA(-, \{\bot \leq \top\})$. The pair $(Q, F)$ is our leading example.

(ii) When $\mathcal{X}$ is the category of posets, the upset functor $U: \text{Pos} \to \text{DLat}$ mapping a poset to the distributive lattice of upsets is dual to the prime filter functor, naturally isomorphic to $\text{DLat}(-, \{\bot \leq \top\})$.

(iii) The contravariant functor $S: \text{Meas} \to \text{MSL}$ mentioned above is dual to the filter functor $\mathcal{F}$ which maps a semi-lattice $A$ to its collection of filters with the $\sigma$-algebra generated by the units $(\eta(a))_{a \in A}$.

A wide range of state-based systems can be formulated as Set coalgebras, and beyond Set we have further examples, including a) descriptive Kripke frames as Stone coalgebras of the Vietoris topology [16]; b) positive Kripke frames as Pos.
coalgebras of the convex powerset functor [13]; and c) Markov processes as \textit{Meas} coalgebras of the Giry monad [9].

2.2 Factorisation systems

\textbf{Definition 2.3} (see [3]) Given two classes $E, M$ of morphisms of a category $\mathcal{X}$, we say that $(E, M)$ is a factorisation system if

(i) every morphism $f$ has an $(E, M)$-factorisation;

(ii) $E$ and $M$ contain all isomorphisms and are closed under composition.

(iii) it has the diagonal fill-in property; i.e., for each equation $g \circ e = m \circ f$ with $e \in E$ and $m \in M$ there is a unique morphism $d$ such that the equations $d \circ e = f$ and $m \circ d = g$ hold.

We say that a factorisation system $(E, M)$ is proper if $E$ is contained in the class of epimorphisms and $M$ in the class of monomorphisms.

\textbf{Proposition 2.4} For any factorisation system $(E, M)$ on $\mathcal{X}$, the following statements are true:

(i) $E$-morphisms are preserved by pushouts;

(ii) $M$ is closed under limits in the arrow category $\mathcal{X}^\to$;

(iii) if $f \circ g$ and $f$ are $M$-morphisms, then so is $g$.

2.3 Equationally presentable functors

Let $\mathcal{A}$ be a finitary variety with a left adjoint $F$ to the forgetful functor $U: \mathcal{A} \rightarrow \text{Set}$, and $J$ the full inclusion of the subcategory $\mathcal{A}_f \omega$ of $\mathcal{A}$ on free algebras $F\eta$ for $\eta \in \mathbb{N}$.

\textbf{Definition 2.5} (see [4, 29, 6]) An endofunctor $L$ of $\mathcal{A}$ is finitely based if it satisfies one of the following equivalent statements:

(i) $L$ is finitary and preserves canonical presentations;

(ii) $L$ is of the form $L \times J L J$, a left Kan extension of $L J$ along $J$.

Let $\text{FinB}[\mathcal{A}, \mathcal{A}]$ denote the full subcategory of the functor category $[\mathcal{A}, \mathcal{A}]$ on finitely based functors.

The notion of finitely based functors plays an important role in our theory of one-step semantics, since they are precisely the equationally presentable ones. Some elementary facts are given first.

\textbf{Proposition 2.6} $\text{FinB}[\mathcal{A}, \mathcal{A}]$ is equivalent to $[\mathcal{A}_f \omega, \mathcal{A}]$.

Every variety $\mathcal{A}$ is locally presentable and (effectively) regular so the same holds for $\text{FinB}[\mathcal{A}, \mathcal{A}]$ by this equivalence and $\mathcal{A}_f \omega$ being small. Another useful fact about $\text{FinB}[\mathcal{A}, \mathcal{A}]$ is that every functor has a finitely based coreflection:

\textbf{Lemma 2.7} $\text{FinB}[\mathcal{A}, \mathcal{A}]$ is a coreflective subcategory of $[\mathcal{A}, \mathcal{A}]$ with the right adjoint $L \times J (\circ J)$. 

115
Chen and Jung

It follows that $\text{FinB}[\mathcal{A}, \mathcal{A}]$ is closed under colimits of $[\mathcal{A}, \mathcal{A}]$ since the inclusion is a left adjoint. Moreover, it can be shown further that the coreflector sends finite products to $\text{FinB}[\mathcal{A}, \mathcal{A}]$ unchanged:

**Proposition 2.8** $\text{FinB}[\mathcal{A}, \mathcal{A}]$ is closed under finite products of $[\mathcal{A}, \mathcal{A}]$.

The finitely based coreflexion of $L$ is written as $\rho_L: \hat{L}_\omega \Rightarrow L$ with $\hat{L}_\omega = \text{Lan}_J LJ$.

The slogan ‘every finitely based functor is equationally presentable’ is justified as follows. A **signature** is a functor $\Sigma$ from the discrete category $\mathbb{N}$ of natural numbers to $\text{Set}$. Every signature defines a finitely based endofunctor on $\mathcal{A}$ by $\hat{H}_\Sigma \Sigma \ldots = \bigcup_{n \in \mathbb{N}} \Sigma_n \times X^n$.

**Theorem 2.9** The functor $\text{FinB}[\mathcal{A}, \mathcal{A}] \to [\mathbb{N}, \text{Set}]$ defined by mapping $L$ to $(n \mapsto ULF)$, is finitary and monadic with a left adjoint defined by (1).

By monadicity, every finitely based functor $L$ has a canonical presentation, i.e. a coequaliser $\hat{H}_\Gamma \Rightarrow \hat{H}_\Sigma \Rightarrow L$ where $\Sigma = ULF$ and $\Gamma = U \hat{H}_\Sigma F$ are signatures. By adjunction, the two parallel morphisms correspond to $\Gamma \Rightarrow U \hat{H}_\Sigma F = U F H\Sigma UF$, (2) so $H\Sigma$ and the parallel morphisms are the generator and the equation of the presentation, respectively.

**Remark 2.10** For $n \in \mathbb{N}$, an element $t$ of $U \hat{H}_\Sigma F n$ is to be understood as a term in $\mathcal{A}$ consisting of at most one layer of operations in $\Sigma$ at terms of $n$ variables, called a **rank-1 $\Sigma$-term**: $U \hat{H}_\Sigma F n = U F \Sigma \text{-operations } \bigcup_{n \text{-ary terms}} F_n$

The two parallel morphisms (2) can be presented by a family of sets $E_n \subseteq (U \hat{H}_\Sigma F n)^2$ of **rank-1 equations**, indexed by $n \in \mathbb{N}$.

**Example 2.11** Modal algebras can be characterised as algebras for a finitely based functor $M$, defined by the following $\text{BA}$-presentation $^4$

$$\text{MA} := \text{BA} \langle \{ \diamond a \}_{a \in A} | \diamond \bot = \bot, \diamond (a \lor b) = \diamond a \lor \diamond b \rangle$$

and $Mf(\diamond a) := \diamond fa$ for each homomorphism $f$. This functor has an equational presentation consisting of a signature $\Sigma_1 := \{ \diamond \}$ and $\Sigma_n := \emptyset$ otherwise; and equations $E_n$ by

$$E_n := \begin{cases} \{ \diamond \bot \sim \bot \} & \text{if } n = 0 \\ \{ \diamond (a \lor b) \sim \diamond a \lor \diamond b \} & \text{if } n = 2 \\ \emptyset & \text{otherwise} \end{cases}$$

$^4$ A $\text{BA}$-presentation $\text{BA} \langle \mathcal{G} \ | \ E \rangle$ indicates an algebra generated by $\mathcal{G}$ subject to equations in $E$. 

116
where 2 is presented as \{a, b\}.

**Remark 2.12** In categories such as \textbf{Set} and the category \textbf{Vec}_K of vector spaces over a field \(K\), every finitely generated algebra is free, so finitary functors are finitely based. Moreover, every finitary endofunctor of \(\textbf{BA}\) is naturally isomorphic to a finitely based functor on non-trivial Boolean algebras [20]. However, this coincidence does not hold in general, and a counterexample can be found in [29, Example 3.12].

### 3 Categories of One-Step Semantics

We study properties of the category of all one-step semantics, including colimits, limits, a monoidal structure, coreflections, and its opposite category. For each type \(T\) of coalgebras, we explore the category of one-step semantics of \(T\) and show that there is always a ‘full’ one-step semantics to which every one-step semantics can be translated in a unique way.

\[ \mathcal{P} : X \rightarrow \mathcal{A} \] always denotes a contravariant functor. It is suppressed if an ambiguity is unlikely.

#### 3.1 The category of all one-step semantics

**Definition 3.1** A \textbf{one-step semantics over} \(P\) consists of an endofunctor \(T\) of \(X\), called the \textbf{type of one-step semantics}; an endofunctor \(L\) of \(A\), called the \textbf{syntax of modalities}; a natural transformation \(\delta : LP \Rightarrow PT\), called the \textbf{interpretation of modalities}, denoted by \((L, T, \delta)\).

**Proposition 3.2** Every one-step semantics \((L, T, \delta)\) defines a functor \(P^\delta\) from the category of \(T\)-coalgebras to the category of \(L\)-algebras, called the \textbf{lifting of} \(P\).

**Example 3.3** (Classical modal logic) Kripke semantics for modal logic with its algebraic semantics defines a one-step semantics \((M, P, \delta)\) over \(Q\) as follows where \(M\) is given in (3) and \(P\) is the covariant powerset functor.

Define a natural transformation \(\diamond : 2^\rightarrow \Rightarrow 2^P\) by \(S \mapsto \{ U \in PX \mid U \cap S \neq \emptyset \}\) for each subset \(S \subseteq X\). Then, \(\diamond\) defines a natural transformation \(\delta\) from \(MQ\) to \(QP\) by mapping \(\diamond S\) to \(\diamond_X S\), since \(\diamond_X\) satisfies the above two equations in (3) for every \(X\).

The lifting given by the one-step semantics \((M, P, \delta)\) is the functor mapping every \(P\)-coalgebra \(\xi : X \rightarrow PX\) to the \(M\)-algebra \(Q\xi \circ \delta_X : MQX \rightarrow QX\) which is equivalent to the \textbf{complex algebra} of \((X, \xi)\). The unique algebra homomorphism \([\cdot] : \Phi \rightarrow QX\) from the initial \(M\)-algebra \((\Phi, \alpha)\) to the complex algebra interprets every element in \(\Phi\) to a predicate on \(X\). The semantics of possibility, \([\diamond \varphi] = \{ x \in X \mid \xi(x) \cap [\varphi] \neq \emptyset \}\), follows from the commutativity of homomorphisms.

Polyadic predicate liftings for a \textbf{Set} functor \(T\), i.e. natural transformations \(\lambda : (2^\rightarrow)^n \Rightarrow 2^T\) for the contravariant powerset functor \(2^\rightarrow\), also provide a class of examples:
Example 3.4 (Predicate liftings) Every set \( \Lambda \) of polyadic predicate liftings for \( T \) gives rise to a one-step semantics \((L^\Lambda, T, \delta^\Lambda)\) over \( Q \) as follows. Let \( \Lambda_n \) denote the set of \( n \)-ary predicate liftings in \( \Lambda \) so that \( \Lambda \) is a signature \( \Lambda : \mathbb{N} \to \text{Set} \) and \( F \) the left adjoint to the forgetful functor \( U : \text{BA} \to \text{Set} \). Define the syntax functor \( L^\Lambda \) by the signature \( \Lambda \) without equations, and the interpretation \( \delta^\Lambda : L^\Lambda Q \Rightarrow Q T \) on generators by \((\lambda, S) \mapsto \lambda X(S)\) for \( \lambda \in \Lambda_n \) and each \( n \)-tuple \( S = (S_i \subseteq X)_{i \in n} \).

Example 3.5 (Cover modality [18]) The cover modality for a finitary and weak-pullback preserving \( \text{Set} \) functor \( T \) with Boolean logic also defines a one-step semantics. Let \( \in_X \) denote the membership relation on a set \( X \). Define a function \( \nabla^T_X : T^2 X \to 2^{TX} \) by

\[
\alpha \mapsto \{ t \in TX \mid (t, \alpha) \in \overline{T}(\in_X) \}
\]

where \((t, \alpha) \in \overline{T}(\in_X)\) if there is \( w \in T(\in_X) \) with \( T \pi_1(w) = t \) and \( T \pi_2(w) = \alpha \). \( \nabla^T_X \) is natural in \( X \) because \( T \) preserves weak pullbacks. By adjunction and \( 2^T = U\overline{Q} \), the transpose of \( \nabla^T \) is a natural transformation \( \overline{\nabla^T} \) from \( FTU \overline{Q} \) to \( QT \). That is, the cover modality defines a one-step semantics \((FTU, T, \overline{\nabla^T})\).

Definition 3.6 The category \( \text{CoLog}^P \) of one-step semantics over \( P \) is defined to be the comma category \((P^* \downarrow P^*)\) from the precomposition \( P^* : L \mapsto \overline{LP} \) to the postcomposition \( P^* : T \mapsto \overline{PT} \). (Following our convention, we will usually suppress the superscript \( P \).)

That is, the objects of \( \text{CoLog} \) are one-step semantics \((L, T, \delta)\) and a morphism from \((L, T, \delta)\) to \((L', T', \delta')\) is a pair of natural transformations \((\tau : L \Rightarrow L', \nu : T' \Rightarrow T)\) satisfying \( P\nu \circ \delta = \delta' \circ \tau P \). The natural transformation \( \tau \) is intuitively understood as a translation from syntax \( L \) to syntax \( L' \). We will justify this intuition in Section 4. We denote the projection \((L, T, \delta) \mapsto L \) with \( U_L \) and the contravariant \((L, T, \delta) \mapsto T \) with \( U_R \).

Proposition 3.7 (Coreflection) Given a coreflective subcategory \( \mathcal{B} \) of the functor category \([\mathcal{A}, \mathcal{A}]\), the pullback of the full inclusion functor \( J : \mathcal{B} \to [\mathcal{A}, \mathcal{A}] \) along the forgetful functor \( U_L : \text{CoLog} \to [\mathcal{A}, \mathcal{A}] \) is also coreflective.

Note that the pullback category is the full subcategory of \( \text{CoLog} \) of those one-step semantics whose syntax functor lies in \( \mathcal{B} \).

Proof sketch Let \( \rho_L : L^\uparrow \Rightarrow L \) denote the coreflection of the functor \( L \). Then, the coreflection of a one-step semantics \((L, T, \delta)\) is \((L^\uparrow, T, \delta \circ \rho_L)\). \( \square \)

3.1.1 Colimits and limits

A colimit of a \( J \)-indexed diagram \( D \) in \( \text{CoLog} \) can be constructed by a pointwise colimit \((\tau_i : L_i \Rightarrow L)_{i \in J}\) of \( U_LD \) and a pointwise limit \((\nu : T \Rightarrow T_i)_{i \in J}\) of \((U_RT)_{i \in J}\).
using the universal property:

\[
\begin{array}{c}
L_i P \\ (L_i P + (U_L Df)_P) \\ L_j P \quad \delta_i \\
\end{array}
\]

\[
\begin{array}{c}
(\delta_i \circ \pi_i) \\ \delta_j \\
\end{array}
\]

since by assumption \(LPX\) is a colimit for each component \(X\).

**Theorem 3.8 (Colimit)** The pair \((U_L, U_R)\) of projections of \text{CoLog} creates pointwise colimits.

A pointwise colimit means that it is constructed by a pointwise colimit in \([\mathcal{A}, \mathcal{A}]\) and a pointwise limit in \([\mathcal{X}, \mathcal{X}]\).

**Example 3.9** (Multi-modal logic) For a set \(A\) of labels, the \(A\)-fold coproduct of classical modal logic \((M, \mathcal{P}, \delta)\) is a multi-modal logic for \(A\)-labelled Kripke frames.

Limits in \text{CoLog} can be constructed similarly as

\[
\text{Lim}L_i P \xrightarrow{\delta = \text{Lim}\delta_i} \text{Lim}PT_i \cong P(\text{Colim}T_i)
\]

and \(\text{Lim}\delta_i\) is a pointwise limit in the arrow category, provided that \(P\) maps a colimit to a limit:

**Theorem 3.10 (Limit)** Suppose that \(P\) has a dual adjoint \(S\). Then, the pair \((U_L, U_R)\) of projections creates pointwise limits.

**Example 3.11** An alternating system over a set \(A\) of actions \([10]\) is a coalgebra for the functor \(D + \mathcal{P}^A\) (where \(D\) is the probability distribution functor). For such systems, a one-step semantics can be obtained as a product of a one-step semantics of type \(\mathcal{P}^A\) and one of type \(D\). For the former we may take multi-modal logic, and the latter can be probabilistic modal logic induced by predicate liftings \((p)\) for \(D\), indexed by \(p \in \mathbb{Q} \cap [0,1]:: \)

\[
S \mapsto \{ \mu \in \mathcal{D}X \mid \sum \mu(S) \geq p \}
\]

for each subset \(S \subseteq X\).

### 3.1.2 Composition

The composition \(\otimes\) of one-step semantics is defined by pasting:

\[
\begin{array}{c}
L_1 \\
L_1 \otimes L_2 \\
L_1 L_2 \\
\end{array}
\]

\[
\begin{array}{c}
T_1 \\
T_1 T_2 \\
T_1 T_2 \\
\end{array}
\]

119
Lemma 3.12 The composition $\otimes$ of one-step semantics is a bifunctor, mapping each pair of morphisms $(\tau_o, \nu_o): \delta_1 \to \delta_3$ and $(\tau_e, \nu_e): \delta_2 \to \delta_4$ to a morphism from $\delta_1 \otimes \delta_2$ to $\delta_3 \otimes \delta_4$ and defined by the horizontal composites

$$
\begin{array}{ccc}
L_2 & \longrightarrow & L_1 \\
\downarrow \tau_e & & \downarrow \tau_o \\
L_4 & \longrightarrow & L_3
\end{array}
\quad
\begin{array}{ccc}
T_2^{op} & \longrightarrow & T_1^{op} \\
\downarrow \nu_e & & \downarrow \nu_o \\
T_4^{op} & \longrightarrow & T_3^{op}
\end{array}
$$

Theorem 3.13 (Monoidal structure) The composition $\otimes$ of one-step semantics with the identity semantics $(I, I, \text{id}_P)$ is a strict monoidal structure on $\text{CoLog}$.

Example 3.14 A simple Segala system over a set $A$ of actions [28] is a coalgebra for the composite $\mathbb{P}^A \circ D$. Thus, a one-step semantics for simple Segala systems can be obtained as the composite of the $A$-fold coproduct of $(M, \mathbb{P}, \delta)$ and the one-step semantics of probabilistic modal logic.

3.1.3 Mate correspondence

To finish our study on $\text{CoLog}$, we study the mate correspondence of one-step semantics, a tool used first by Klin [14] to analyse one-step expressiveness of coalgebraic logic in a categorical approach:

Definition 3.15 Suppose that $P$ has a dual adjoint $S$. The mate $\delta^*$ of a natural transformation $\delta: LP \Rightarrow PT$ is a natural transformation from $TS$ to $SL$ defined by the pasting diagram

$$
\begin{array}{ccc}
T & \longrightarrow & L \\
\downarrow \eta & & \downarrow \delta \\
S & \longrightarrow & P
\end{array}
\quad
\begin{array}{ccc}
P & \longrightarrow & \delta P \\
\downarrow \psi & & \downarrow \epsilon \\
S^{op} & \longrightarrow & T^{op}
\end{array}
$$

in the opposite of $\mathcal{X}$, that is, $\delta^* = SL\eta \circ S\delta S \circ \epsilon TS$.

The mate operation maps a one-step semantics to an object of the comma category $\text{CoLog}^* := (S^* \downarrow S_*)$. Furthermore:

Proposition 3.16 $\text{CoLog}$ is dually isomorphic to $\text{CoLog}^*$ where a one-step semantics $(L, T, \delta)$ is mapped to its mate $(T, L, \delta^*)$.

Remark 3.17 By this isomorphism, colimits and limits in $\text{CoLog}$ are transposed to their duals in $\text{CoLog}^*$, but more specifically, it can be shown that every (pointwise) colimit in $\text{CoLog}$ can be constructed as a pointwise limit in the arrow category, to be used in Theorem 5.10. A bifunctor $\oplus$ defined by $\theta_1 \oplus \theta_2 := \theta_1 T_2 \circ T_1 \theta_2$ with the identity $\text{id}: S \Rightarrow S$ defines a strict monoidal structure on $\text{CoLog}^*$, which is the image of $(\otimes, \text{id})$ under the mate correspondence.

3.2 Fibre categories of $\text{CoLog}$

Definition 3.18 For every endofunctor $T$ of $\mathcal{X}$, the category $\text{CoLog}_T$ is defined to be the fibre category over $T$. More precisely, the objects of $\text{CoLog}_T$ are one-step
semantics of type $T$, denoted $(L, \delta)$. A morphism $\tau$ from $(L, \delta)$ to $(L', \delta')$ is a natural transformation satisfying $\delta = \delta' \circ \tau P$ (i.e. a syntax translation).

The type functor being fixed, we focus on the syntax projection functor which maps $U_L: (L, \delta) \mapsto L$ and $\tau \mapsto \tau$.

**Proposition 3.19** The following statements hold:

(i) $U_L$ reflects isomorphisms.

(ii) For every coreflective subcategory $\mathcal{C}$ of the functor category $[\mathcal{A}, \mathcal{A}]$, the pullback of the full inclusion $i: \mathcal{C} \hookrightarrow [\mathcal{A}, \mathcal{A}]$ along $U_L: \text{CoLog}_T \rightarrow [\mathcal{A}, \mathcal{A}]$ is coreflective.

(iii) $U_L$ creates pointwise colimits.

We are ready to establish a fundamental theorem for coalgebraic modal logics representable as one-step semantics:

**Theorem 3.20** Suppose that $P$ has a dual adjoint $S$. Then every fibre $\text{CoLog}_T$ has a terminal object $(PTS, PT\epsilon: PTS P \Rightarrow PT)$ where $\epsilon: I \rightarrow SP$ is the counit of the dual adjunction.

In every fibre category $\text{CoLog}_T$, this terminal object is called the **full one-step semantics** for $T$. It may be too elusive for practical purposes, but conceptually it explains that every collection of modalities may be viewed as a (uniquely determined) fragment of this canonical one-step semantics.

### 4 Equational One-Step Semantics

We now focus on one-step semantics whose syntax functor is defined by operations and equations, called *equational*. To work with equational one-step semantics, we use finitely based functors (as introduced in Section 2) as syntax functors. Equational one-step semantics are characterised as (generalised) predicate liftings subject to equations. In particular, a full equational one-step semantics exists and is the logic of all predicate liftings subject to a complete axiomatisation *up to isomorphism*.

#### 4.1 The category of equational one-step semantics

**Definition 4.1** A one-step semantics $(L: \mathcal{A} \rightarrow \mathcal{A}, T, \delta)$ is (finitary) **equational** if $\mathcal{A}$ is a variety and $L$ is finitely based. $\text{ECoLog}$ is the corresponding full subcategory of $\text{CoLog}$.

The examples given in Section 3 are equational, since $\text{BA}$ is a variety and $M, L^\Lambda$ for a set $\Lambda$ of predicate liftings, and $L^{\nabla T}$ for a finitary and weak-pullback preserving functor $T$, are all finitely based.

It is not hard to show that the composite $L_1 \circ L_2$ of finitely based functors remains finitely based, so every composite of equational one-step semantics remains equational.
Proposition 4.2 The composition \( \otimes \) on \( \mathsf{ECoLog} \) with the identity semantics \((I, I, \text{id}_P)\) is a strict monoidal structure on \( \mathsf{ECoLog} \).

By applying Proposition 3.7 to the finitely based coreflection \( \rho_L : \hat{L}_\omega \Rightarrow L \), every one-step semantics has an equational coreflection:

Proposition 4.3 \( \mathsf{ECoLog} \) is a coreflexive subcategory of \( \mathsf{CoLog} \). Therefore, \( \mathsf{ECoLog} \) is closed under colimits of \( \mathsf{CoLog} \).

As we are actually interested in equational one-step semantics rather than all one-step semantics, the coreflection ensures that colimits are still constructed as in \( \mathsf{CoLog} \). By Proposition 2.8, this is also true for finite products:

Proposition 4.4 Suppose that \( P \) has a dual adjoint. Pointwise finite products in \( \mathsf{ECoLog} \) coincide with products in \( \mathsf{CoLog} \).

Corollary 4.5 The category of equational one-step semantics over the contravariant \( 2^-, Q, U, \) and \( S \) in Example 2.2, respectively, has colimits and finite products constructed pointwise.

For example, the one-step semantics for alternating systems in Example 3.11 is indeed a product in \( \mathsf{ECoLog} \).

The universal property of (co)limits hints at certain optimal conditions. For instance, the fusion of predicate liftings [8], known as the smallest conservative extension of two given logics of predicate liftings, is in fact the coproduct in \( \mathsf{ECoLog} \) of the corresponding equational one-step semantics:

Example 4.6 (Binary coproduct as fusion) Given two sets \( \Lambda_1 \) and \( \Lambda_2 \) of polyadic predicate liftings for \( \mathsf{Set} \) functors \( T_1 \) and \( T_2 \), respectively, the coproduct of the one-step semantics induced by \((\Lambda_i)_{i=1,2}\) consists of \( T_1 \times T_2 \), as its type, and \( L := L^{\Lambda_1} + L^{\Lambda_2} \cong \tilde{H}_\Lambda \) with \( \Lambda := \Lambda_1 + \Lambda_2 \), as its syntax, and as its interpretation the natural transformation \( \delta : LQ \Rightarrow QT \) defined for each set \( X \) on the generators of \( L \) by

\[
(\lambda, S) \mapsto (\pi_1^{-1} \circ \lambda_X)(S)
\]

for each \( \lambda \in \Lambda_{i,n} \) and \( n \)-tuple \( S = (S_j)_{j \in n} \).

4.2 Fibre categories of \( \mathsf{ECoLog} \)

Definition 4.7 For any endofunctor \( T \) of \( \mathcal{X} \), \( \mathsf{ECoLog}_T \) is defined to be the fibre category of \( \mathsf{ECoLog} \) over \( T \).

By Proposition 3.19, each \( \mathsf{ECoLog}_T \) is a coreflective subcategory of \( \mathsf{CoLog}_T \). Now the equational version of Theorem 3.20 follows, as the coreflector preserves limits, including the terminal object:

Corollary 4.8 Suppose that \( P \) has a dual adjoint \( S \). Every fibre \( \mathsf{ECoLog}_T \) has a terminal object

\[
(\hat{P}TS_\omega, \hat{P}T\epsilon \circ \rho_P : \hat{P}TS_\omega P \rightarrow PT)
\]
where \( \rho \) is the finitely based coreflection \( \overline{PTS}_\omega \Rightarrow PTS \) and \( \epsilon \) is the counit of the dual adjunction.

Again, this terminal object is called the full equational one-step semantics for \( T \), and every equational one-step semantics for \( T \) is a fragment of it.

The remaining part of this section is used to describe equational one-step semantics as logics of predicate liftings subject to rank-1 equations.

### 4.2.1 Predicate liftings

**Definition 4.9** Let \( U : \mathcal{A} \to \text{Set} \) be a functor and \( T \) an endofunctor of \( \mathcal{A} \). A (finitary) predicate lifting for \( T \) is a natural transformation from \( (UP)^n \) to \( UPT \) where \( n \in \mathbb{N} \).

When \( P = Q \), this definition boils down to the usual definition of polyadic predicate liftings, since the underlying set \( UQX \) is the powerset \( 2^X \).

**Example 4.10** When \( P = S : \text{Meas} \to \text{MSL} \), a predicate lifting for \( T \) maps a measurable set of a measurable space \( X = (X,S_X) \) to a measurable set on \( TX \) natural in \( X \). Take the Giry monad \( G \), for example; \( G \) maps a space \( (X,S_X) \) to the collection of subprobability distributions \( \mu : S_X \to [0,1] \) satisfying

\[
\mu(\emptyset) = 0 \quad \text{and} \quad \mu\left(\bigcup_i M_i\right) = \sum_i \mu(M_i)
\]

for countable unions of pairwise disjoint measurable sets \( M_i \) (the \( \sigma \)-algebra on \( GX \) is ignored). Then a predicate lifting for \( G \) can be defined for each \( p \in [0,1] \) by

\[
M \mapsto \{ \mu \in GX : \mu(M) \geq p \}
\]

for \( M \in S_X \), which is exactly the modality in [11, Section 4.3].

Assume that \( \mathcal{A} \) is a variety and \( F \) denotes the left adjoint to the forgetful functor \( U \). Every \( n \)-ary predicate lifting \( \lambda \) for \( T \) determines a one-step semantics \( (FU^n, \lambda^*) \), called the unimodal logic of \( \lambda \), where \( \lambda^* : F(UP)^n \Rightarrow PT \) is the transpose of \( \lambda \) by adjunction. A set \( \Lambda \) of predicate liftings for \( T \) determines a one-step semantics, called the logic of predicate liftings, consisting of

\[
\widehat{H}_\Lambda = F \left( \prod_{n \in \mathbb{N}} \Lambda_n \times (U-)^n \right) \quad \text{and} \quad \delta_X^\Lambda : \widehat{H}_\Lambda PX \Rightarrow PTX,
\]

mapping generators \( (\lambda,S) \) to \( \lambda_X(S) \) for \( \lambda \in \Lambda_n \) and \( S = (S_i \in UPX)_{i \in n} \).

Moreover, every finitely based functor \( L \) is a coequaliser of parallel morphisms \( \widehat{H}_\varepsilon \cong \widehat{H}_\Sigma \Rightarrow L \), so characterisations follow readily:

**Corollary 4.11** For every endofunctor \( T \) of \( \mathcal{A} \),

(i) every logic of predicate liftings for \( T \) is a coproduct of unimodal logics;

(ii) every object in \( \mathbb{ECoLog}_T \) is a coequaliser of a logic of predicate liftings.
4.2.2 Translations between equational one-step semantics

By Corollary 4.11, a morphism from $(L',\delta')$ to $(L,\delta)$ in ECologT boils down to a family of translations from a unimodal logic to $(L,\delta)$, indexed by some set $\Lambda$ of predicate liftings $\lambda$:

$$F(UPX)^n \xrightarrow{\tau_X} LPX$$

Commutativity implies that a translation is not only a syntactic translation but also preserves the interpretation.

**Example 4.12** (Continuing Example 3.5) Possibility $\Diamond$ and necessity $\Box$ can be translated to the cover modality $\nabla$ by setting

$$\Box \varphi := \nabla\{\varphi\} \lor \nabla\emptyset \quad \text{and} \quad \Diamond \varphi := \nabla\{\varphi, \top\}.$$ 

The syntactic translation defines a morphism $\tau$ from the one-step semantics $(M,\delta)$ to the one-step semantics $(FP\omega U, \nabla P\omega)$ of the cover modality, i.e. $\nabla \circ \tau Q = \delta$.

4.2.3 Equations valid under an interpretation

Let $\Lambda$ be a set of predicate liftings. A rank-1 $\Lambda$-term in $n$ variables can be interpreted by a function that maps any $n$-tuple of predicates on $X$ to a predicate on $TX$ as follows. By adjunction, every $n$-tuple $S = (S_i \in PX)$ is presented as a morphism $(Fn \xrightarrow{a} PX)$. Define the interpretation $\llbracket - \rrbracket^S_{\Lambda,X}$ for terms with $n$ variables as the composite

$$U\widehat{H}_\Lambda Fn \xrightarrow{U\widehat{H}_\Lambda a} U\widehat{H}_\Lambda PX \xrightarrow{U\delta^\Lambda} UPTX$$

where $\delta^\Lambda_X$ is the logic of predicate liftings in $\Lambda$.

**Definition 4.13** Given a set $\Lambda$ of predicate liftings for $T$, a rank-1 equation $t \sim t'$ of $\Lambda$ is valid under the interpretation of $\Lambda$ if $\llbracket t \rrbracket^S_{\Lambda,X} = \llbracket t' \rrbracket^S_{\Lambda,X}$ for any $X$ and $n$-tuple $(S_i \in UPX)$.

The universal quantifiers can be simplified to a pair of parallel morphisms:

**Theorem 4.14** Given a set $\Lambda$ of predicate liftings for $T$ and families of rank-1 $\Lambda$-equations $E_n$ with $n$ variables indexed by $n \in \mathbb{N}$, the following statements are equivalent:

(i) For each $n$, every equation $t \sim t' \in E_n$ is valid under the interpretation of $\Lambda$.

(ii) The following diagram commutes

$$\widehat{H}_\xi P \xrightarrow{\pi_1P} \widehat{H}_\Lambda P \xrightarrow{\delta^\Lambda} PT$$

---

Footnote 5: For consistency, we restrict to the finitary powerset functor $\mathcal{P}_\omega$, otherwise $FP\omega U$ is not finitely based.
where \( \pi_1, \pi_2 : \widehat{H_E} \Rightarrow \widehat{H_A} \) are the parallel morphisms induced by \( E = (E_n)_{n \in \mathbb{N}} \).

Therefore, a regular quotient of a one-step semantics in \( ECoLog_T \) retains the set of modalities but they are subject to more equations. Moreover, every translation factors through a regular quotient:

**Theorem 4.15** For every \( T \), the category \( ECoLog_T \) has \((\text{RegEpi}, U^{-1}\text{Mono})\) as a factorisation system where \( U \) is the forgetful functor \( U : ECoLog_T \rightarrow \text{FinB}[\mathcal{A}, \mathcal{A}] \).

This factorisation system also leads us to an important notion:

**Definition 4.16** An equational one-step semantics for \( T \) is said to have a **complete axiomatisation** if it has no proper regular quotient in \( ECoLog_T \).

Informally, a one-step semantics \((L, \delta)\) has a complete axiomatisation if every rank-1 equation valid under the interpretation \( \delta \) is derivable from the presentation of \( L \). Note that this is not model-theoretic completeness.

**Example 4.17** The one-step semantics \((M, \delta)\) of classical modal logic has a complete axiomatisation. Any proper regular quotient of \( M \) would identify a rank-1 equation which is not derivable from \( \Diamond \bot = \bot \) and \( \Diamond(a \lor b) = \Diamond a \lor \Diamond b \). However, classical modal logic is complete with respect to the class of all Kripke frames, so there exists an instance refuting the equation, a contradiction. Thus \((M, \delta)\) has a complete axiomatisation.

### 4.2.4 Objects of predicate liftings

Using the Yoneda Lemma, Schöder observes \([26]\) that \( n \)-ary predicate liftings in the case where \( P = Q \) are in bijection with the subsets of \( T2^n \), because \( UQ = 2^\bot \) is naturally isomorphic to \( \text{hom}(\cdot, 2) \). We generalise and combine this with Klin’s *objects of \( T \)-modalities* \([14]\), again employing the Yoneda Lemma and the dual adjunction:

**Lemma 4.18** Suppose that \( P \) has a dual adjoint \( S \) and \( \mathcal{A} \) is a variety with a left adjoint \( F \) to the forgetful functor \( U \). For any endofunctor \( T \) of \( \mathcal{A} \), the following statements hold:

(i) For any \( n \in \mathbb{N} \), the set \( U(PTS)Fn \) is in bijection with the collection of \( n \)-ary predicate liftings.

(ii) The bijection is natural in objects \( n \) in the Kleisli category of the induced monad \( UF \).

That is, \( PTSFn \) is precisely the **object of \( n \)-ary predicate liftings**. This ensures that the collection of all finitary predicate liftings is small, so a coproduct of all unimodal logics exists. Naturality means that for any function \( f : n \rightarrow UFm \) between
free algebras, the diagram

\[
\begin{array}{ccc}
UPTSFn & \xrightarrow{[-]} & \text{Nat}(UP^n, UPT) \\
\downarrow UPTSf & & \downarrow f^* \\
UPTSFm & \xrightarrow{[-]} & \text{Nat}(UP^m, UPT)
\end{array}
\]

commutes, where \([-]\) indicates the bijection from \(UPTSFn\) to the collection of \(n\)-ary predicate liftings, \(\bar{f}\) is the transpose of \(f\), and \(\hat{f}^*\) is pre-composition with \(\hat{f}\):

\[
\hat{f} : (m \xrightarrow{a} UP) \mapsto (n \xrightarrow{f} UFm \xrightarrow{ua} UFUP \xrightarrow{Up} UP).
\]

**Remark 4.19** Note that the forgetful functor \(U : \mathcal{A} \to \text{Set}\) is naturally isomorphic to \(\text{hom}(F1, -)\) by adjunction, so the composite \(UP\) is naturally isomorphic to \(\text{hom}(-, SF1)\) by dual adjunction. Let \(\Omega\) be \(SF1\). Then, an \(n\)-ary predicate lifting is a natural transformation from \(\text{hom}(-, \Omega)^n\) to \(\text{hom}(T-, \Omega)\), which is in a more familiar form. Specifically, the underlying set of \(SF1\) is a two-element set of truth values for the dual adjoints in Example 2.2.

### 4.2.5 Characterising the full equational one-step semantics

The finitely based part \(\hat{PTS}_\omega\) of \(PTS\) has a canonical presentation, so the full equational one-step semantics is a regular quotient of the one-step semantics consisting of

\[
F \left( \coprod_{n \in \mathbb{N}} UPTSFn \times U^n \right) \quad \text{and} \quad (PTe \circ \rho P) \circ e
\]

where \(\rho : \hat{PTS}_\omega \Rightarrow PTS\) is the coreflection and \(e\) the regular quotient. We call (4) the logic of all predicate liftings for \(T\).

**Lemma 4.20** The following statements hold:

(i) The logic of all predicate liftings is a coproduct of all unimodal logics.

(ii) The full equational one-step semantics has a complete axiomatisation.

**Theorem 4.21** An equational one-step semantics \((L, \delta)\) of type \(T\) is isomorphic to the full equational one-step semantics iff \((L, \delta)\) has a complete axiomatisation and every unimodal logic has a translation to \((L, \delta)\).

**Proof sketch** By the previous lemmas, the ‘only if’ part follows. For the converse, let \((L, \delta)\) be a one-step semantics with a complete axiomatisation such that every unimodal logic has a translation to it. By assumption, there exists a mediating morphism \(\tau\) from the logic \((L^A, \delta^A)\) of all predicate liftings to it, since \((L^A, \delta^A)\) is a coproduct of all unimodal logics. Consider the pushout of \(\tau\) and the regular quotient
Using the factorisation system on $ECoLog_T$ and the fact that the full semantics is terminal, the statement follows.

\section{Logic of a One-Step Semantics}

We remind the reader how a one-step semantics provides a coalgebraic modal logic, and refine Klin’s expressiveness condition \cite{Klin} for finitary functors. We also address the modularity problem of expressiveness.

\begin{itemize}
  \item For the sake of brevity, we restrict our discussion to the case of a one-step semantics which has a language (see below). Also, we assume that $P$ has a dual adjoint $S$.
\end{itemize}

\subsection{Logical setup}

Definition 5.1 The language of $(L, T, \delta)$ is the initial $L$-algebra, denoted $(\Phi_L, \alpha_L)$.

The initial $L$-algebra can be constructed by the initial sequence \cite{Adamek}. For a finitary functor $L$, the following $\omega$-sequence starting from the initial object $0$:

$$0 \xrightarrow{\lambda} L0 \xrightarrow{L} \cdots \xrightarrow{L} L^\omega 0 \xrightarrow{L} \cdots$$

has a colimit $(f_i: L^i 0 \rightarrow L^\omega 0)_{i \in \omega}$. Then, by $L$ being finitary, it is not hard to conclude that there exists an isomorphism $\beta: L^\omega 0 \rightarrow LL^\omega 0$, so we can set $\Phi_L := L^\omega 0$ and $\alpha_L := \beta^{-1}$ for the initial $L$-algebra.

Definition 5.2 The semantic interpretation $[\cdot](X, \xi)$ of a language $(\Phi_L, \alpha_L)$ in a $T$-coalgebra $(X, \xi)$ is the unique $L$-algebra homomorphism from $(\Phi_L, \alpha_L)$ to the $L$-algebra $(PX, P\xi \circ \delta)$.

The logic of $(L, T, \delta)$ refers to the language of $L$ and its semantic interpretation. The interpretation $[\cdot]$ maps a formula $\varphi$, as an element of $\Phi$, to the subset of $X$ consisting of those states that satisfy $\varphi$. For example, if the syntax functor is induced by a set $\lambda \in \Lambda$ of (unary) predicate liftings, then $\alpha(\lambda, \varphi)$ represents the modal formula $\lambda \varphi$. The semantics $[\lambda \varphi]$ on a $T$-coalgebra $(X, \xi)$ is given by the diagram

\begin{center}
\begin{tikzcd}
L\Phi \arrow[r, \alpha] \arrow[d, L[-]] & \Phi \arrow[d, [\cdot]] \\
PX \arrow[r, \delta_x] & PTX \arrow[r, P\xi] & PX
\end{tikzcd}
\end{center}
so that \([\lambda \phi] = (P_\xi \circ \delta_X \circ L[-])(\lambda, \phi) = P_\xi(\lambda_X [\phi])\). By definition, soundness is easy to see, since any rank-1 equation encoded in the syntax functor \(L\) is valid in the language \((\Phi_L, \alpha_L)\).

**Definition 5.3** The theory map for a \(T\)-coalgebra \((X, \xi)\) is the transpose \(th: X \to S\Phi\) of the semantic interpretation \([-]\) under the dual adjunction \(S \dashv P\).

Intuitively, the theory map simply maps every state \(x\) to the collection of formulae satisfied by \(x\), which is indeed the case for, say, the dual adjoints in Example 2.2.

Assuming that \(\mathcal{X}\) has kernel pairs, define logical equivalence of a theory map \(th_\xi\) to be the kernel of \(th_\xi\); a logic of \((L, T, \delta)\) may then be said to be expressive if the logical equivalence is contained in some kernel of a coalgebra homomorphism regarded as an \(\mathcal{X}\)-morphism. For concrete categories, such as \(\text{Set}\) and \(\text{Meas}\), two elements \(x\) and \(y\) of a coalgebra \((X, \xi)\) are logically equivalent if \(th_\xi(x) = th_\xi(y)\); two elements are behaviourally equivalent if there exists a coalgebra homomorphism \(f\) with \(fx = fy\), and thus a logic is expressive if logically equivalent elements are also behaviourally equivalent.

It is known that behavioural equivalence implies logical equivalence, i.e., adequacy holds. The converse, expressiveness, is more interesting, and we turn to it next.

### 5.2 Expressiveness

We have shown that every equational one-step semantics has a unique translation to the full equational one-step semantics (Corollary 4.8), which is in fact the most expressive logic, so its expressiveness is equivalent to the existence of an expressive logic of some one-step semantics:

**Theorem 5.4** Every morphism in \(\text{CoLog}_T\) preserves expressiveness.

Recall Klin’s general condition for expressiveness in the functorial framework (in the formulation by Jacobs and Sokolova [11]):

**Theorem 5.5** (see [14,11]) Suppose that \(\mathcal{X}\) has a proper factorisation system \((\mathcal{E}, \mathcal{M})\). Then, if \(a)\ T\ preserves \(\mathcal{M}\)-morphisms and \(b)\ the mate \(\delta^*\) is a pointwise \(\mathcal{M}\)-morphism, then \(th(x) = th(y)\) implies that \(x\) and \(y\) are behaviourally equivalent.

Note that this result only gives sufficient conditions for expressiveness, but on the positive side, these conditions are particularly suitable for further generalisation. Hence, in the presence of a proper factorisation system, we define an equational one-step semantics of \(T\) to be one-step expressive if\(a\) \(T\) preserves \(\mathcal{M}\)-morphisms, and \(b\) the mate \(\delta^*\) is a pointwise \(\mathcal{M}\)-morphism.

### 5.3 Expressiveness for finitary functors

We restrict attention to strongly locally finitely presentable categories:

---

\(^6\) The justification of this point-free formulation may be found in [6, Section 4.1.5] (it assumes the existence of a proper factorisation system). In the following we will simply assume that \(\mathcal{X}\) is concrete though this does not necessarily imply the existence of kernel pairs.
Definition 5.6 (see [2]) A locally finitely presentable category is strongly locally finitely presentable if for every cofiltered limit \((\sigma_i: Y \to Y_i)_{i \in I}\) and every monomorphism \(f: X \into Y\) with \(X\) finitely generated, there is \(i \in I\) such that the composite \(\sigma_i \circ f\) is monic.

For example, \(\text{Set}, \text{Pos},\) and \(\text{Vec}_K\) are strongly locally finitely presentable. Klin showed [14] that when \(\mathcal{A}\) is a locally finitely presentable category the full finitary one-step semantics\(^7\) of a finitary functor on a strongly locally presentable category is one-step expressive if the counit \(\epsilon: I \to S\mathcal{P}\) is pointwise monic. We adapt Klin’s argument and apply it to the full equational one-step semantics in the case that \(\mathcal{A}\) is a variety. By Theorem 5.4, the equational version also recovers the finitary one.

Theorem 5.7 (c.f. [14, Theorem 4.4]) Let \(\mathcal{X}\) be a strongly locally presentable category, \(\mathcal{A}\) a variety, and \(T: \mathcal{X} \to \mathcal{X}\) a finitary and monomorphism-preserving functor. If the counit \(\epsilon: I \to S\mathcal{P}\) is pointwise monic, then the full equational one-step semantics of \(T\) is one-step expressive.

Proof sketch Klin’s theorem for full finitary one-step semantics is established in two steps. Let \(\mathcal{A}_\omega\) be the full (small) subcategory on finitely presentable objects. First it is shown that if for every \(A\) the source

\[
\{TSf: TSA \to TSA_1\}_{f \in (\mathcal{A}_\omega \downarrow A)} \quad (5)
\]

is jointly monic, then the mate \(\delta^*\) is pointwise monic where \((\mathcal{A}_\omega \downarrow A)\) is the comma category from \(\mathcal{A}_\omega\) to \(A\). Second, the family (5) is shown to be jointly monic by the strong local presentability.

For the same reason, to show that \(\delta^*\) is pointwise monic, it suffices to show that the source

\[
\{TSG: TSA \to TSFn\}_{g \in (\mathcal{A}_\omega' \downarrow A)} \quad (6)
\]

is jointly monic. However, every morphism \((Fn \xrightarrow{g} A)\) factors through a regular epimorphism \(e: Fn \twoheadrightarrow B\) with \(B\) finitely presentable, and \(TSe\) is a monomorphism by the dual adjunction and assumption. It is easy to see that (6) is jointly monic if and only if (5) is jointly monic. Then Klin’s second step completes this proof. \(\square\)

5.4 Modularity of one-step expressiveness

As we discussed colimits, finite products, and compositions on \(\text{CoLog}\), it is of interest to know if one-step expressiveness is stable under these constructions at this level of generality. Surprisingly, compositions and colimits preserve one-step expressiveness in a straightforward way:

Theorem 5.8 The composite \(\delta_1 \otimes \delta_2\) of one-step expressive semantics \((L_1, T_1, \delta_1)\) and \((L_2, T_2, \delta_2)\) is one-step expressive for \(T_1T_2\).

\(^7\) This terminology is defined analogously: A one-step semantics is finitary if its syntax functor is finitary on a locally finitely presentable category. The full finitary one-step semantics for \(T\) is the terminal object in the category of finitary one-step semantics for \(T\) by Proposition 3.19.

129
Proof By Remark 3.17, the mate of $\delta_1 \otimes \delta_2$ is equal to $\delta_1^* L_2 \circ T_1 \delta_2^*$. By assumption, $T_i$ preserves $\mathcal{M}$-morphisms and $\delta_i^*$ is a pointwise $\mathcal{M}$-morphism for $i = 1, 2$. Hence the composite $\delta_i^* L_2 \circ T_1 \delta_2^*$ is a pointwise $\mathcal{M}$-morphism. By assumption, $T_1 T_2$ preserves $\mathcal{M}$-morphisms.

Example 5.9 The double finite powerset functor $\mathbb{P}_\omega \circ \mathbb{P}_\omega$ does not have a separating set of unary predicate liftings [26]. However, we may simply self-compose the usual one-step semantics for $\mathbb{P}_\omega$ to obtain a one-step expressive logic.

Theorem 5.10 The pointwise colimit of one-step expressive semantics is one-step expressive.

Proof Let $D$ be a diagram in $\text{CoLog}$. The colimit of $D$ is a one-step semantics of type $\text{Lim}_i T_i$. By assumption, each $T_i$ preserves $\mathcal{M}$-morphisms and by Proposition 3.19, $\mathcal{M}$-morphisms are closed under limits in $\mathcal{X} \rightarrow$, so $\text{Lim}_i T_i$ preserves $\mathcal{M}$-morphisms.

The mate of $(\text{Colim} D)$ is a pointwise $\mathcal{M}$-morphism: Every $D_i^*$ is a pointwise $\mathcal{M}$-morphism by assumption, so the limit of $D_i^*$ in the arrow category $\mathcal{X} \rightarrow$ is also a pointwise $\mathcal{M}$-morphism. By Remark 3.17, $\text{Lim}(D_i^*)$ is isomorphic to $(\text{Colim} D)^*$ and the latter is a pointwise $\mathcal{M}$-morphism since $\mathcal{M}$ contains isomorphisms.

Example 5.11 (Labelling $T^A$) Suppose that $\mathcal{X}$ and $\mathcal{A}$ have products and coproducts, respectively. Let $A$ be a set of labels. Every coalgebra $\xi : X \rightarrow T^A X$ for the $A$-fold product of $T$ corresponds to a family $(\xi_a)_{a \in A}$ of $T$-coalgebras, i.e. an $A$-labelled $T$-coalgebra, and the $A$-fold coproduct of a one-step semantics $(L, T, \delta)$ defines a one-step semantics for $T^A$. Moreover, the coproduct is one-step expressive if and only if $(L, T, \delta)$ is one-step expressive. The result applies immediately to $\mathbb{P}$, $\mathcal{D}$, the convex powerset functor $\hat{\mathbb{P}}$, and the Giry monad $\mathcal{G}$, to name but a few.

As for finite products, we are encouraged by the result of Cîrstea’s and Pattinson [7] that one-step expressiveness is preserved by finite products for one-step semantics over $2^\rightarrow$, but we do not have a general proof at this point.

Acknowledgement

We would like to thank Paul Levy, Clemens Kupke and Alexander Kurz for insightful discussions and feedback on the current work.

References


Total Maps of Turing Categories

J.R.B. Cockett\textsuperscript{1,2}

\textit{Department of Computer Science}

\textit{University of Calgary}

Calgary, Alberta, Canada, T2N 1N4

P.J.W. Hofstra\textsuperscript{3}

\textit{Department of Mathematics and Statistics}

\textit{University of Ottawa}

Ottawa, Ontario, Canada, K1N 6N5

P. Hrubeš\textsuperscript{4}

\textit{Department of Computer Science and Engineering}

\textit{University of Washington}

Seattle, WA, USA

Abstract

We give a complete characterization of those categories which can arise as the subcategory of total maps of a Turing category. A Turing category provides an abstract categorical setting for studying computability: its (partial) maps may be described, equivalently, as the computable maps of a partial combinatory algebra. The characterization, thus, tells one what categories can be the total functions for partial combinatory algebras. It also provides a particularly easy criterion for determining whether functions, belonging to a given complexity class, can be viewed as the class of total computable functions for some abstract notion of computability.

Keywords: Computability theory, Partial Combinatory Algebra, Turing Category, Complexity Theory.

1 Introduction

Turing categories \cite{1,2,6}, provide an abstract categorical setting for computability theory which, unlike partial combinatory algebras (PCAs) and related structures, are presentation-independent and purely formulated in terms of categorical properties. The standard example of a Turing category has objects powers of the natural
numbers and maps all the partial recursive functions. However, there are many other less well-known examples deriving from the “computable maps” of PCAs (or more generally relative PCAs) or from syntactical methods. Of special relevance for the purposes of the present work are the Turing categories described in [3] which have as total maps the programs belonging to various complexity classes (PTIME, LOGSPACE, etc.). These examples naturally lead to the question of exactly what categories can be the total maps of a Turing category. Intuitively, as in any Turing category one can simulate all computable functions, it would seem reasonable to suppose that the total maps would have to satisfy some fairly demanding closure properties.

The question is of significance for various reasons. To start with, it is one way to determine the limits of the applicability of Turing categories in studying computability. If it were impossible for the total maps of a Turing category to be exactly, say, the linear time functions, then one cannot hope to use Turing categories as a basis for investigating feasible computation at very low complexity levels. On the other hand, if one knows that the total maps of a Turing category can be of such low complexity then Turing categories can be a tool for formally unifying computability and complexity theory and allowing a fluid flow of ideas between the subjects.

A second reason for considering this question in the abstract categorical setting is that it leads to an interesting comparison with the more traditional view on these matters, namely via logic. Given a logical theory, one may ask which relations are provably total: the weaker the theory, the smaller the class of provably total maps. One may also wish for the system to be strong enough to allow for the representation of partial computable maps. It is well known that even relatively weak fragments of arithmetic ensure this. For example, Robinson’s Arithmetic Q is enough to ensure all the partial recursive functions are represented. The study of complexity, using bounded and two sorted logics [5] for example, has further pushed the limits of these methods. It is, thus, in its own right, an interesting question to know exactly what an absolutely minimal logic for generating these settings really is. Although, this paper does not attempt to answer this question directly, the categorical framework we describe can certainly be backward engineered into a logical form: even a brief perusal indicates that there is a significant difference between these approaches, not least because a Turing category is not a priori based on arithmetic.

The third point of interest lies not so much in the question itself as in the methods used here to provide the answer. The proof that a cartesian category satisfying certain conditions can be embedded as a category of total maps in a Turing category makes use of two ideas: first, it uses the Yoneda embedding to create a canonical category of partial maps into which the original category embeds. Next, we use the concept of a stack machine in the presheaf category to create a partial combinatory algebra which in turn will generate the desired Turing category. A stack machine can be thought of as a categorical implementation of the canonical rewrite system in combinatory logic (but augmented with additional data). As such, this concept helps clarify the connection between syntactical approaches to
generating models of computation and categorical methods.

Below we develop necessary and sufficient conditions for a Cartesian category to be the total maps of a Turing category. The conditions are perhaps a little surprising as apparently very little is actually required: there must be a universal object, $U$, which has a pair of “disjoint” elements, and which has an “abstract retract” structure which allows coding of maps. A universal object is an object into which every other object can be embedded as a subobject: to have such an object is already a somewhat non-trivial requirement as this, in particular, implies there is an embedding $U \times U \to U$ showing that $U$ must be an infinite object. The remaining conditions are somewhat more technical: they are explained in section 3 below. However, it should be noted that in most of the standard applications, as described in section 5, these technical conditions can be by-passed.

We start the exposition by considering the much more general question of how a category can arise as the subcategory of total maps of an arbitrary restriction category. (For background on restriction categories we refer to [4].) This leads to the notion of a totalizing extension of the given category, and we show that the category of totalizing extensions of a particular category has a final object which is naturally a restriction category. This insight allows us to transfer the general question of finding a Turing category which extends a given cartesian category into finding a partial combinatory algebra in this final extension whose total maps include the given maps. This perspective allows us to propose necessary conditions for a Cartesian category to be the total maps of a Turing category, see section 3. To show that these are sufficient we demonstrate that one can build, using a simple abstract machine, a combinatory algebra in the final totalizing extension (actually of a slightly modified category) which has elements representing all the total maps: this suffices in view of Theorem 4.12 of [2]. Finally we provide simpler sufficient conditions to show the wide applicability of the theorem.

2 Totalizing map subcategories

When $X$ is a restriction category, the inclusion of the subcategory of total maps $\text{Total}(X) \to X$ satisfies various properties. This leads to the following definition:

**Definition 2.1** [Totalizing functor] A functor $T : X' \to X$ is totalizing when it satisfies the following three conditions:

(i) $T$ on objects, $T_{\text{Obj}} : \text{Obj}(X') \to \text{Obj}(X)$, is an isomorphism,

(ii) $T$ is faithful,

(iii) $T$ is left factor\(^5\) closed, meaning that when $Th = gf$ then there is a (necessarily unique) $k$ such that $Tk = f$.

The following observation indicates that this class of functors is reasonably well-behaved.

---

\(^5\) Note that “left” factor refers to the diagrammatic order of composition: $A \xrightarrow{Th} B \xrightarrow{f} B \xrightarrow{g} C$. 

134
Lemma 2.2 In the category of categories and functors, the class of totalizing functors, $\mathcal{T}$, form a stable system of monics:

(i) Every $T \in \mathcal{T}$ is monic,
(ii) $\mathcal{T}$ is closed to composition,
(iii) $\mathcal{T}$ contain all isomorphisms,
(iv) $\mathcal{T}$ is closed to pulling back along any functor.

Proof. Routine. □

Recall that given any (small) category $X$ the Yoneda embedding, $y : X \to \text{Set}^{X^{\text{op}}}$, gives a full and faithful embedding of $X$ into presheaves on $X$. This category of presheaves is finitely complete and therefore one can form the partial map category on all monics $\text{Par}(\text{Set}^{X^{\text{op}}})$. Notice that the total maps in this category are exactly the morphisms of $\text{Set}^{X^{\text{op}}}$. Restricting this category to the representables gives a full subcategory which we shall denote $\overline{X}$. While this is in general no longer a partial map category, it certainly is a restriction category. We then have the following pullback:

$$
\begin{array}{ccc}
X & \xrightarrow{\eta} & \overline{X} \\
\downarrow y & & \downarrow \overline{y} \\
\text{Set}^{X^{\text{op}}} & \xrightarrow{\sim} & \text{Par}(\text{Set}^{X^{\text{op}}})
\end{array}
$$

This exhibits $\overline{X}$ as a subcategory of total maps of the restriction category $\overline{X}$, whence $\eta$ is totalizing.

While there may be many other categories $Y$ into which $X$ embeds via a totalizing functor, we shall now make precise the sense in which $\eta : X \to \overline{X}$ is the universal such functor.

Consider the category of totalizing extensions of $X$, denoted $\text{Ext}_{\mathcal{T}}(X)$, whose objects are totalizing functors $T : X \to Y$ and whose morphisms are commuting triangles below $X$

$$
\begin{array}{ccc}
X & \xrightarrow{T} & Y \\
\downarrow T' & & \downarrow F \\
\overline{X} & \xrightarrow{\overline{T'}} & \overline{Y}
\end{array}
$$

which reflect total maps in the sense that the square on the right is a pullback.

Proposition 2.3 $\text{Ext}_{\mathcal{T}}(X)$ is a (finitely) complete category in which $\eta : X \to \overline{X}$ is the final object.

Proof. The verification that pullback exist (and are constructed as in $\text{Cat}$ is routine. We shall show that $\eta : X \to \overline{X}$ is the final object in this category. To this end, suppose $\overline{T} : X \to \overline{Y}$ is totalizing. Without loss of generality we may assume that $Y$ has the same collection of objects as $X$. It is then obvious how the functor $E : Y \to \overline{X}$ should act on objects. For a morphism $g : X \to X'$ in $Y$, we need to define a
partial map $y(X) \to y(X')$. First consider the following sieve on $X$:

$$S_g = \{ h : Z \to X | gh \in X \}.$$ 

This sieve corresponds to a subobject of the representable $y(X)$. Next, define a natural transformation $\tau(g) : S_g \to y(X')$ by

$$\tau(g)_Z : S_g(Z) \to X(Z, X'); \quad \tau(g)_Z(h) = gh.$$ 

This data defines a partial map $E(g) : y(X) \to \check{Y}(X')$, which is total if and only if $g \in X'$. This in particular shows that $E$ reflects total maps. The verification that $E$ is functorial is straightforward and left to the reader.

To show that $E$ is unique suppose we are given an extension $E' : Y \to \check{X}$. Since $E'$ must respect maps from $X$, we verify its action on a map $g : X \to X'$ which is not in $X$. Then $E'(g)$ is a span $y(X) \supset S \to y(X')$. Now if $h \in S_g$, the composite $gh$ is in $X$, and therefore $h \in S$, whence $E'(gh) = E'(g)E'(h) = \sigma(h)$ must equal $y(gh)$. This means that $E'(g) \geq E(g)$. To show the converse, assume that $h \in S$ but $h \notin S_g$. Then $gh \notin X$. But then $E'(gh)$ is a total map, contradicting the reflection of total maps. □

We also note that in case $\check{Y}$ is a restriction category and $T : X \to \check{Y}$ the inclusion of total maps, then in fact $E : \check{Y} \to \check{X}$ is a restriction functor. In addition, we have the following result, which states that if $X$ has products, then $E$ preserves the induced restriction products.

**Lemma 2.4** Suppose that $X$ has finite products. Then $\check{X}$ also has finite restriction products, and $\eta$ preserves them.

**Proof.** Finite products in a split restriction category are completely determined by their counterparts in the total map subcategory. Thus, as the inclusion $\text{Set}^{X^\text{op}} \to \text{Par}(\text{Set}^{X^\text{op}})$ preserves products and $y : X \to \text{Set}^{X^\text{op}}$ preserves products cutting down to $\check{X}$ preserves products. □

With a view towards our aim of characterizing those categories which arise as the total map subcategory of a Turing category, we can now observe the following. When $X \to T$ is a totalizing extension with $T$ a Turing category, then we have the following situation:

![Diagram]

Then by the above lemma, $E$ is a Cartesian restriction functor; since any such functor preserves partial combinatory algebras, we find that there is a PCA $E(U)$, the image of the Turing object $U \in T$. Thus we obtain:

**Proposition 2.5** $X$ is the total map category of a Turing category if and only if there is a combinatory algebra in $\check{X}$ whose total computable maps include the maps of $X$. 

136
Thus we may see our original problem as one of finding a suitable PCA in $\tilde{X}$ for which all the maps in $X$ are computable. Indeed, if such a PCA exists, then we may let $T$ be the subcategory of $\tilde{X}$ on the computable maps. Notice one rather nice aspect of this reformulation of the problem: since the maps of $X$ already account for all the total maps in $\tilde{X}$, the total maps represented by a PCA in $\tilde{X}$ necessarily lie in $X$ already; thus we don’t have to worry about having too many total maps represented. Therefore, the only thing to verify when constructing a candidate PCA is that it represents all the maps in $\tilde{X}$.

3 Properties of the total maps of a Turing category

We are now in a position to collect the necessary conditions for being the total maps of a Turing category. Clearly the total map category has to be Cartesian (i.e. has finite products), as Turing categories are Cartesian by definition. The remaining conditions are somewhat more technical. However, it should be stressed that in many cases of interest these conditions greatly simplify.

The purpose of the section is to prove:

**Proposition 3.1** Every total map category of a Turing category has a universal object which has a pair of disjoint elements and is equipped with an abstract retract structure for which there exist codings.

Below we introduce the required notions of having a universal object, disjoint elements, abstract retract structure, and codes, showing each is present in the total map category of a Turing category.

### 3.1 A universal object

One of the properties of the Turing object $U$ in a Turing category is that every object is a retract of it. More explicitly, given any object $A$, there are morphisms $\iota_A : A \to U$ and $\rho_A : U \to A$ for which $\rho_A \iota_A = 1_A$. This forces $\iota_A$ to be a total map (in fact, a monomorphism), but $\rho_A$ can still be partial. We will typically denote the situation by $A \prec U$, or by $(\iota_A, \rho_A) : A \prec U$ if necessary.

We call an object $U$ in a category universal if for every object $A$ there exists a monomorphism $\iota_A : A \to U$. (Thus we don’t ask that every object is a retract of $U$; this is stronger, and will be analyzed in the section on abstract retract structures below.)

**Lemma 3.2** In any Cartesian category $\mathbb{X}$ with a universal object $U$:

(i) There is always an element $\iota_1 : 1 \to U$ of the universal object;

(ii) There is always an embedding $\iota_{U \times U} : U \times U \to U$;

(iii) The homset $\mathbb{X}(U, U)$ either has one element, in which case $\mathbb{X}$ is trivial, or is infinite.
3.2 A pair of disjoint elements

A Turing object $U$ in a Turing category admits the interpretation of all combinatory logic terms. In particular, $U$ has two global elements $t = \lambda^* xy.x$ and $f = \lambda^* xy.y$. We shall prove that these two elements have the property of being disjoint in the sense that whenever $th = fh$ for a map $h$, the domain $\bar{h}$ of $h$ must be a strict initial object in the idempotent splitting. For the subcategory of total maps, this simply means that for any $h$ with $th = fh$ the domain of $h$ is a strict initial object.

**Proposition 3.3** In every Turing category the total maps have a universal object $U$ with a disjoint pair of elements.

Notice first that when the Turing category is trivial (in the sense that it is equivalent to the terminal category) all objects are strict initial objects, and hence all elements are disjoint. In general, in a Turing category, an element will be disjoint from itself only when the category is trivial as this forces the initial and final object to be the same.

We start by observing:

**Lemma 3.4** Let $X$ be a Cartesian category. A map $h : H \to 1$ makes

$$
H \times A \xrightarrow{h \times 1} 1 \times A \xrightarrow{\pi_1} A \xrightarrow{f} B
$$

commute for all $f, g : A \to B$ if and only if $H$ is a strict preinitial object.

Recall that an object $H$ is called preinitial if there is at most one map from that object to any other object. It is strict preinitial if any object with a map to $H$ is itself (strict) preinitial. Preinitial objects are quite common: for example in the category of (commutative) rings $\mathbb{Z}$ is the initial object, while $\mathbb{Z}_n$ is preinitial for each $n$. However, neither $\mathbb{Z}$ nor $\mathbb{Z}_n$ are strict preinitial.

**Proof.** Suppose $x, y : H \to A$. Then:

$$
x = x\pi_1(h, 1) = x\pi_1(h \times 1)\Delta = y\pi_1(h \times 1)\Delta = y\pi_1(h, 1) = y.
$$

So $H$ is preinitial. To show that it is strict, suppose that $q : Q \to H$; then the above reasoning applies to $qh$, making $Q$ preinitial. \qed

Notice that if $H$ is strict preinitial then, as $H \times Y \xrightarrow{\pi_0} H$, it follows that $H \times Y$ is always strict preinitial. This means in a Cartesian category once one has one strict preinitial there must, for each object, be a preinitial with a map to that object. This does not imply there will be an initial object, but it does force that an initial object, if it exists, is automatically strict.

We are now ready to prove that the elements $t$ and $f$ in a Turing category are disjoint.
Lemma 3.5 In any Turing category, if \( h : H \to 1 \) is a total map which equalizes \( t \) and \( f \) (as chosen above) then \( H \) is a strict initial object in the total map category. Thus the elements \( t \) and \( f \) are disjoint.

Proof. Suppose \( f, g : A \to B \) are total maps. Without loss of generality we may assume \( A = B = U \). Note that the diagram

\[
\begin{array}{ccc}
1 \times U & \xrightarrow{\pi} & U \\
t \times \langle f, g \rangle & & g \\
U \times U \times U & \xrightarrow{\bullet \times 1} & U
\end{array}
\]

commutes serially. But then precomposing with \( h \times 1 : H \times U \to 1 \times U \) shows that \( h \) satisfies the conditions of the lemma above and so is a strict preinitial object. It remains only to show that there is a total map \( H \to B \) for each object \( B \). Each \( B \) is a retract of the Turing object and so the composite \( \iota_B \rho_B : U \to U \) is a split idempotent. As with any map in a Turing category, there is a code \( k_B : 1 \to U \) such that \( \bullet(k_B, 1) = \iota_B \rho_B \), as in

\[
\begin{array}{ccc}
U & \xrightarrow{\rho_B} & B \\
& \xrightarrow{\bullet(k_B, 1)} & U
\end{array}
\]

While \( \rho_B \) may be partial, we argue that \( \rho_B h \) is total:

\[
\bullet(k_B, 1)h = \bullet(k_Bh, h) = \bullet(kh, h) = \bullet(k \times 1) \Delta h = \pi_0 \Delta h = h
\]

so that \( \iota_B \rho_B h \) is total, whence \( \rho_B h \) is total.

It may be useful at this stage to provide an example of a total map category which has a universal object and yet cannot be the total maps of a Turing category. The simplest example, which also shows that such a category cannot consist entirely of preinitial objects, is when the category is a meet-semilattice. Then the only element is the identity on the top and this must be disjoint from itself, forcing the top to also be the bottom thereby collapsing the lattice.

3.3 An abstract retract structure

Recall that each object \( A \) in a Turing category comes equipped with \( (\iota_A, \rho_A) : A \prec U \) exhibiting it as a retract of the (chosen) Turing object. There may be many choices for this family of retractions although we shall assume that \( (\iota_U, \rho_U) = (1_U, 1_U) \). Below we describe how this structure introduces an analogous structure on the total map category.

First, consider a span between representable objects \( y(X) \) and \( y(X') \) in \( \text{Set}^{X^{\text{op}}} \):

\[
y(X) \xleftarrow{s} A \xrightarrow{t} y(X').
\]
The apex \( A \) need not be representable, but we may consider the family of spans in \( X \) arising by covering \( A \) by representables as follows:

\[
\mathcal{S}(s, t) = \{ X \leftarrow B \xrightarrow{th} y(X') | h : Y(B) \to A \}.
\]

This family is then clearly closed under precomposition with maps in \( X \). In fact, we may regard \( \mathcal{S}(s, t) \) as a category whose objects are the spans in \( X \) factoring through \((s, t)\), and whose morphisms are morphisms of spans.

Conversely, consider a family \( \mathcal{R} \) of spans in \( X \) from \( X \) to \( X' \) which is closed under precomposition. For the present purposes, we shall call such a family an abstract span. Regarding \( \mathcal{R} \) as a category, we may consider the functor

\[
\mathcal{R} \to \text{Span}(y(X), y(X')) \xrightarrow{\text{Apex}} \text{Set}^{X^{\text{op}}}
\]

which sends a span \( X \leftarrow B \xrightarrow{q} X' \) to the \( y(B) \). The colimit of this diagram gives a span \( y(X) \leftarrow \hat{\mathcal{R}} \to y(X') \) in \( \text{Set}^{X^{\text{op}}} \). The proof of the following is now routine:

**Lemma 3.6** The assignments \((s, t) \mapsto \mathcal{S}(s, t)\) and \(\mathcal{R} \mapsto \hat{\mathcal{R}}\) are mutually inverse (up to isomorphism of spans).

Next, we wish to characterize when the left leg of a span between representables is monic (so that it is a partial map). Call an abstract span \( \mathcal{R} \) deterministic when \((p, q), (p, q') \in \mathcal{R}\) implies \(q = q'\).

**Lemma 3.7** Given an abstract span \( \mathcal{R} \) corresponding to a span \( y(X) \leftarrow \hat{\mathcal{R}} \xrightarrow{\iota} y(X') \) between representables, \( s \) is monic if and only if the spans in \( \mathcal{R} \) are deterministic.

Of course, in this situation \( \hat{\mathcal{R}} \) may be regarded as a sieve on \( X \): it is precisely the sieve

\[
\hat{\mathcal{R}}(Z) = \{ h : Z \to X | (h, k) \in \mathcal{R} \text{ for some } k : Z \to X' \}.
\]

So far, we have described partial maps between representables in terms of abstract spans. Next, we wish to characterize when such a partial map is in fact a retraction of a morphism \( \iota_A : A \to U \). So suppose that in \( \text{Set}^{X^{\text{op}}} \), we have a span \( y(X) \supseteq S \xrightarrow{\iota} y(X') \) which is a retraction of a map \( \iota : X' \to X \). This of course means that \( \iota \in S \) and \( t(\iota) = 1_{X'} \). In terms of the corresponding abstract span \( \mathcal{S} \), this means that \((\iota, 1) \in \mathcal{S}\) and that for each \((h, f) \in \mathcal{S}\) and each diagram

\[
\begin{array}{ccc}
X' & \xleftarrow{k} & P \\
\downarrow{\iota} & & \downarrow{z} \\
X & \xleftarrow{h} & M \xrightarrow{f} X'
\end{array}
\]

with the square commuting, we have \( fz = k \). We shall refer to this condition by saying that the abstract span \( \mathcal{S} \) is **retracting** on \( \iota \). To summarize:
Lemma 3.8 Given a morphism \( \iota : X' \rightarrow X \) in \( \mathbb{X} \) and a span \( y(X) \supseteq S \rightarrow y(X') \), the span is a retraction of \( \iota \) in \( \text{Par}(\mathbb{X}^{\text{op}}) \) if and only if the corresponding abstract span contains \((\iota, 1)\) and is retracting on \( \iota \).

We are now ready for the main definition of this section:

Definition 3.9 An abstract retract structure on a Cartesian category with a universal object \( U \) consists of a choice of embeddings \( \iota_A : A \rightarrow U \) (with \( \iota_U = 1_U \)) and, in addition, for each object \( A \) an abstract retraction of \( \iota_A \), meaning a family of spans \( \mathcal{R}_A \) from \( U \) to \( A \) satisfying:

[RS.1] Each \( \mathcal{R}_A \) is closed under precomposition;
[RS.2] Each \( \mathcal{R}_A \) is deterministic;
[RS.3] \( U \xleftarrow{\iota_A} A \xrightarrow{1_A} A \in \mathcal{R}_A; \)
[RS.4] Each \( \mathcal{R}_A \) is retracting on \( \iota_A \).

We shall indicate an abstract retraction pair by \((\iota_A, \mathcal{R}_A) : A \prec U\).

Here are some examples abstract retractions for a fixed morphism \( \iota_A : A \rightarrow U \).

Examples 3.1 (1) If \( \mathcal{R} \) is an abstract retraction, then \( \mathcal{R}_U \) is always just the family \( U \xleftarrow{x} X \xrightarrow{x} U \). Indeed suppose \( U \xleftarrow{h} Y \xrightarrow{f} U \in \mathcal{R}_U \) then by [RS.4] \( h1_Y = \iota_U h \), so \( f = f1_X = h \). Similarly, if \( \iota_A \) is an isomorphism then \( \mathcal{R}_A \) is always just the family \( U \xleftarrow{\iota_A} X \xrightarrow{x} A \).

(2) The smallest retraction structure for each \( A \) is the one generated by the span \((\iota_A, 1_A)\). Each span in the abstract retraction is of the form \( U \xleftarrow{\iota_A} X \xrightarrow{x} A \); this means the set is deterministic and retracting on \( \iota_A \).

(3) Suppose that \( \iota_A \) has a (total) retraction \( \rho_A \). Then we may generate a retraction structure \( \{(x, x\rho_A) | x : X \rightarrow U\} \). Note that when \( x = \iota_A \) we obtain the span \( U \xleftarrow{\iota_A} A \xrightarrow{1_A} A \). As all the spans have their right leg determined by the left leg post-composed with \( \rho_A \) the set is deterministic.

(4) Finally, consider a totalizing extension \( T : \mathbb{X} \rightarrow \mathbb{Y} \), in which \( U \) is a universal object. If, in \( \mathbb{Y} \), each object \( A \) is equipped with a retraction \((\iota_A, \rho_A) : A \prec U\), then we may consider the abstract span in \( \mathbb{X} \) given by \( \mathcal{R}_A = \{U \xleftarrow{h} A \xrightarrow{f} A | \rho_A h = f\} \). This is readily seen to be an abstract retraction on \( \iota_A \).

The last example shows in particular that whenever \( T \) is a Turing category with total map subcategory \( \mathbb{X} \), there is an induced abstract retract structure on \( \mathbb{X} \). For the record:

Lemma 3.10 Every Turing category induces, for any Turing object \( U \) and chosen retractions \((\iota_A, \rho_A) : A \prec U\), an abstract retract structure on its total map category.

3.4 Codes

A Turing object \( U \) in \( T \) is not only universal, but it also acts as a weak exponential for every pair of objects in the category. In particular, given \( f : A \rightarrow B \), there
exists a code \( c[f] : 1 \to U \) making
\[
\begin{array}{ccc}
U \times U & \bullet & U \\
\downarrow{(c[f],1)} & & \downarrow{\iota_B} \\
U & \rho_A \rightarrow & A \rightarrow B
\end{array}
\]

commute. Here, \( \rho_A \) and \( \iota_B \) are part of the chosen retractions \( A \prec U \) and \( B \prec U \), and the morphism \( U \times U \rightarrow U \) is the universal application map. The code \( c[f] \) is required to be total, but not unique. Choosing a code \( c[f] \) for each \( f \) amounts to giving a family of mappings

\[ c = c_{A,B} : \mathcal{T}(A, B) \to \mathcal{T}(1, U). \]  

Since the inclusion \( \iota_B \) is total, it in fact suffices to specify codes only for the case \( B = U \), that is, for maps \( A \to U \).

We will now translate this existence of codes to structure on the total maps. Since codes are formulated in terms of the retractions \( \rho_A \), this involves dealing with the abstract retract structure on the total maps. We may cut down the mapping (1) to the total maps to obtain a mapping \( c = c_A : \mathcal{X}(A, U) \to \mathcal{X}(1, U) \), where \( \mathcal{X} = \text{Total}(\mathcal{T}) \).

To motivate the coming definition, first we need a bit of notation. Given a span \( U \leftarrow h X \to k A \) we denote by \( Z^*(h, k) \) the span \( U \leftarrow h \pi_X Z \times X \to k \pi_X A \), obtained by precomposing each leg with the projection \( Z \times X \to X \). When \( \mathcal{R} \) is a family of spans from \( U \) to \( A \) we will write \( Z^* \mathcal{R} = \{(Z^*(h, k))|(h, k) \in \mathcal{R}\} \). Similarly, given \( f : A \to U \), we write \( f_* \mathcal{R} = \{(h, f k)|(h, k) \in \mathcal{R}\} \).

Now consider two maps \( f : A \to U \) and \( g : B \to U \) with codes \( c[f], c[g] \), and suppose we have an object \( Z \) for which the unique total map \( Z \to 1 \) equalizes \( c[f] \) and \( c[g] \). Then as per example 3.1 (4), the a typical span in \( \mathcal{R}_A \) is of the form \( U \leftarrow h X \to k A \) satisfying \( k = \rho_A h \). Given such \( (h, k) \), we may then consider the span
\[
\begin{array}{ccc}
Z \times X & \pi_X \downarrow & \\\nU & \downarrow{h} & X \downarrow{k} & \rightarrow & A \rightarrow U
\end{array}
\]

In the above notation, this is \( Z^*(fh, k) \in Z^* \mathcal{R}_A \). Under the given condition on \( Z \), the composite \( f k \pi_X \) factors as \( g \circ (\rho_B h \pi_X) \), as shown by the following calculation:

\[
\begin{align*}
fk\pi_X &= f\rho_A h \pi_X = f\rho_A \pi_U (z \times h) = \bullet (c_A[f] \times U)(z \times h) \\
&= \bullet (c_A[f]z \times h) = \bullet (c_B[g]z \times h) = \bullet (c_B[g] \times U)(z \times h) \\
&= g\rho_B \pi_U (z \times h) = g\rho_B h \pi_X
\end{align*}
\]

Note that the statement that \( fk\pi_X \) factors as indicated implies that \( Z^*(h, kf) \in Z^* \mathcal{R}_A \) can be regarded as a span of the form \( Z^*(h, qg) \) for \( (h, q) \in \mathcal{R}_B \). By symmetry, this implies that \( Z^* \mathcal{R}_A = Z^* \mathcal{R}_B \). Thus, informally speaking, when
restricting the abstract retractions to \( Z \), the operations of composing with \( f \) and with \( g \) become equal.

This leads to the following:

**Definition 3.11** A Cartesian category \( \mathbb{X} \) with universal object \( U \) and retract structure \((\iota_A, R_A) : A \to U \) has **codes** when there are maps \( c_A : \mathbb{X}(A, U) \to \mathbb{X}(1, U) \) such that whenever \( Z \to 1 \) satisfies \( c[f]z = c[g]z \) for \( f : A \to U, g : B \to U \) then \( Z^* f_* R_A = Z^* g_* R_B \).

The preceding discussion shows:

**Lemma 3.12** The total map subcategory of a Turing category always has codes.

---

### 4 Building a Turing category from total maps

We shall now construct a PCA in \( \widetilde{\mathbb{X}} \) which has all the total maps computable. We use the notion of a “stack object”, which is an object satisfying two domain equations which allow for the implementation of a rewriting system. Using the fact that \( \widetilde{\mathbb{X}} \) admits a trace, we may then define a universal application morphism which will equip the stack object with a PCA structure. From now on, assume \( \mathbb{X} \) is a Cartesian category satisfying the conditions explained in the previous section.

Before we start, however, it is convenient to modify \( \widetilde{\mathbb{X}} \) a little. Recall that the Yoneda embedding does not preserve initial objects (if these happen to be present), but that there is always a subcanonical topology \( J \) on the category \( \mathbb{X} \) which corrects this. (Simply take the smallest topology containing the empty cover on each initial object of \( \mathbb{X} \).) This gives an embedding \( \mathbb{X} \to \text{Sh}(\mathbb{X}, J) \) preserving any existing initial object. For the rest of the paper, we will let \( \overline{\mathbb{X}} \) denote the full subcategory of the partial map category on \( \text{Sh}(\mathbb{X}, J) \) on the representables. It will also be useful to assume that we have access to the splittings of restriction idempotents and to finite coproducts, thus it will be more convenient to work in \( \text{Par}(\text{Sh}(\mathbb{X}, J)) \), and to restricting to \( \overline{\mathbb{X}} \) by observing that the structures we build are always on representable objects.

Another important property of a partial map category of a (pre)sheaf category (and hence also of \( \overline{\mathbb{X}} \)) is that it is traced on the coproduct: given \( f : X \to X + Y \) there is a map \( f^\dagger : X \to Y \) which is the joint of all the finite partial iterates:

\[
f^\dagger = \bigvee_{i \in \mathbb{N}} f^{(i)}, \quad f^{(i)} = \sigma_1^{-1} f \cdots \sigma_0^{-1} f \sigma_0^{-1} f : X \to Y,
\]

where \( \sigma_i^{-1} \) is the partial inverse of the \( i \)-th coproduct injection. Intuitively, \( f^\dagger \) takes an input \( x \in X \), and computes the iterates \( f^\dagger(x) \) for as long as the output lies in \( X \). Once the output lies in \( Y \), this is the value of \( f^\dagger(x) \).
4.1 Stack objects

An object $A$ in a distributive restriction category is said to be a stack object in case there are maps

$$\text{put} : 1 + A \times A \to A \quad \text{get} : A \to 1 + A \times A \quad \text{with} \quad \text{put} \ \text{get} = 1_{1+A \times A}.$$ 

Thus, a stack object can be indicated by $(\text{put}, \text{get}) : 1 + A \times A \prec A$.

**Lemma 4.1** In any distributive restriction category the following are equivalent conditions for an object:

(i) $1 + 1 \prec A$ and $A \times A \prec A$;

(ii) $1 + A \times A \prec A$ (it is a stack object)

(iii) For any polynomial functor $P$, we have $P(A) = n_0.1 + n_1.A + \ldots + n_r.A^r \prec A$

The point is that in $\mathbb{X}$ we know the first condition is satisfied by the universal object $U$: indeed, $U \times U \prec U$ because $U \times U$ is an object of $\mathbb{X}$, and $1 + 1 \prec U$ because $U$ has two disjoint elements $t, f : 1 \to U$, giving $[t, f] : 1 + 1 \to U$. A partial retraction may be defined by letting $S$ be the sieve on $U$ generated by $\{t, f\}$, and by defining a natural transformation $S \sigma \to 1 + 1$ by $\sigma_X(m) = inL(*)$ whenever $m$ factors through $t$, and $inR(*)$ otherwise. Note that this is well-defined precisely because $t, f$ are disjoint, so that $m$ factors through both $t$ and $f$ only when its domain is $0$; but then $(1 + 1)(0)$ is a singleton since $1 + 1$ is assumed to be a sheaf.

4.2 A stack machine for partial combinatory algebras

We shall now use this trace to define a partial application $A \times A \to A$ making $A$ into a PCA. To this end, we will define a partial map $\text{step} : A \times A \times A \to A \times A \times A + A$ whose trace then is of the desired type. More precisely, we will define $x \bullet y := \text{step}^{(x, y, [])}$. The intuition that should be kept in mind is that the map $\text{step}$ executes one step of a program/rewrite system, and that its trace runs the entire computation/rewriting sequence. The components of $A \times A \times A$ will be regarded as a code stack, a value, and a dump stack, respectively. Only when the code is $\text{end}$ and the dump stack is empty does the computation halt.

To define $\text{step}$, we use the domain equations for $A$ to fix three different representations of our stack object:

$$\text{end}, k, s : 1 \to A \quad k_0, s_0 : A \to A \quad s_1 : A \times A \to A \quad \text{nam} : A \to A$$

$$(\langle \text{end}|k|s|k_0|s_0|s_1|\text{nam}, g_0 \rangle : 1 + 1 + 1 + A + A + A \times A + A \prec A$$

$$c_0 : A \to A \quad c_1 : A \times A \to A$$

$$(\langle c_0|c_1 \rangle, g_1) : A + A \times A \prec A$$

$$\text{nil} : 1 \to A \quad \text{cons} : A \times A \to A$$

$$(\langle \text{nil}|\text{cons}, g_2 \rangle : 1 + A \times A \prec A$$

144
and now we define \texttt{step} by the following case distinction:

<table>
<thead>
<tr>
<th>Code</th>
<th>Value</th>
<th>Stack</th>
<th>Code</th>
<th>Value</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>end</td>
<td>(x)</td>
<td>nil</td>
<td>exit with (x)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(k)</td>
<td>(x)</td>
<td>(S)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(k_0(x))</td>
<td>(y)</td>
<td>(S)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(s)</td>
<td>(x)</td>
<td>(S)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(s_0(x))</td>
<td>(y)</td>
<td>(S)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(s_1(x, y))</td>
<td>(z)</td>
<td>(S)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\text{nam}(c_A[f]))</td>
<td>(z)</td>
<td>(S)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>end</td>
<td>(v)</td>
<td>(\text{cons}(c_0(y, z), S))</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>end</td>
<td>(v')</td>
<td>(\text{cons}(c_1(v), S))</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The one aspect which needs explanation concerns how we use the names of maps. The aim is to implement them as the composite of a retraction to the idempotent and the map itself. Without these names, the stack machine can be thought of as an implementation of the usual rewriting system on combinatory logic; adding the names of maps from \(X\) together with the given rule essentially amounts to adding rewrites \(c_A[f] \to f(a)\) to the system.

The \texttt{step} partial function is the join of all its individual components. For \texttt{step} to be well-defined all these components must be compatible, in the sense that they agree on overlaps of domains. Those which are in separate components of the stack object by design are disjoint and so compatible. However, the names of maps are all in the same component and, thus, we must establish compatibility of the implementation of names.

\textbf{Lemma 4.2} \texttt{step} is a well-defined partial map in \(\bar{X}\).

\textbf{Proof.} A typical span corresponding to a partial map \(U \times U \times U \to U \times U \times U\) in the component of \texttt{step} which is defined on a tuple of the form \((\text{nam}(c_A[f]), h, S)\) (for \(f : A \to U, h : X \to U\)) looks like

\[
U \times U \times U \xleftarrow{\text{nam}(c_A[f])} X \xrightarrow{(\text{end}, f, S)} U \times U \times U
\]

Here, \((h, k) \in \mathcal{R}(A)\). Given another such span

\[
U \times U \times U \xleftarrow{\text{nam}(c_B[g])} X' \xrightarrow{(\text{end}, k', S')} U \times U \times U
\]

with \(g : B \to U\) and \((h', k') \in \mathcal{R}_B\), we must show that these two spans are compatible. This is done by considering generalized elements of \(z : Z \to X\) and \(z' : Z \to X'\); we must verify that if \(\langle \text{nam}(c_A[f]) \rangle_{X, h, S} z = \langle \text{nam}(c_B[g]) \rangle_{X', h', S'} z'\), then also \(\langle \text{end} \rangle_{X, f, k, S} z = \langle \text{end} \rangle_{X', g, k', S'} z'\). It is clear that this holds for the last com-
ponent. Thus we must show that if $c_A[f]!_X z = c_B[g]!_{X'} z'$ and $h z = h' z'$ then $f k z = g k' z'$.

The first condition means $c_A[f]!_Z z = c_B[g]!_Z z$ which, by the requirement on codes, implies $(h z, f k z)$ is in $R_B g$. This implies $(h z, f k z) = (h z, g v)$. But $h z = h' z'$, as $R_B$ is deterministic, now gives $v = k' z'$ and so $f k z = g v = g k' z'$ as required. 

We are now ready for the main result:

**Theorem 4.3** Given a Cartesian category $X$ with a universal object, a pair of dis-
joint elements and a retract structure with codes then the above definition of $\bullet$ in $\tilde{X}$
gives a partial combinatory algebra; the subcategory of $\tilde{X}$ on the computable maps is
then a Turing category whose total maps are exactly the maps of $X$.

The proof consists of a verification that the combinators $k$ and $s$ indeed perform
as required. This is relegated to the appendix.

## 5 Applications

For many of the obvious applications we have proven much more than is actually
needed. Here are two corollaries of our main theorem:

**Proposition 5.1** Given a Cartesian category $X$ in which

- Every object has at least one element;
- There is a universal object $U$;
- There is a monic (set) map $c : \bigsqcup_{A \in X} X(A, U) \to X(1, U)$;
- There is a faithful product preserving functor into $U : X \to \text{Set}$.

Then $X$ occurs as the total maps of a Turing category.

These conditions include the PTIME maps between binary natural numbers. In this example one actually can obtain a linear time encoding of pairs of binary number into the binary numbers. This may be achieved by interleaving the bit strings while adding an extra bit on each component to indicate termination. This means we have (much more than):

**Corollary 5.2** The PTIME maps between binary numbers occur as the total maps
of a Turing category.

We also have the following:

**Proposition 5.3** Any countable Cartesian category with a universal object $U$ and
a pair of disjoint elements is the total maps of a Turing category.

Here we observe that once one has two distinct elements the fact that one has
a stack object allows one to easily obtain countably many distinct and disjoint elements. This means that there is a monic assignment of maps to elements and one can use the smallest abstract retractions.
6 Conclusion

To unify the abstract notion of computability embodied in Turing categories with the study of feasible computation (e.g. LINEAR, LOGSPACE, PTIME, ...) minimally one must know whether these functional complexity classes can form the total maps of a Turing category. In [3] it was shown that both LOGSPACE and PTIME functions had a natural description as the total maps of Turing categories. However, in order to obtain those results, it was necessary to use the well-known facts from complexity theory that transducers and Turing machines can universally simulate themselves with an overhead which can be accommodated within (respectively) LOGSPACE and PTIME. This meant the argument that these complexity classes could be modelled by Turing Categories relied heavily on the details of the machine models and the way in which resources were measured. In particular, as there is no widely accepted machine model which can simulate itself with a linear time overhead, the linear time maps, LINEAR, could not be so readily included in this approach.

The power of the results outlined in this paper is, precisely, that they are abstract. That is that they do not depend on the peculiarities of machine models or on the way resources are counted. In particular, Proposition 5.3 applies immediately to the linear time maps because the encoding and decoding of pairs of bit strings can all be done in linear time.

By describing necessary and sufficient conditions for a Cartesian category to be the total maps of a Turing category we have delimited the applicability of Turing categories to feasible computation. Perhaps, somewhat counter-intuitively, the results indicate that Turing categories are applicable beyond the traditional confines of computability theory into feasible computation and the domain of complexity theory. In this regard Turing categories, therefore, provide – by more than mere analogy – a medium for the transfer of ideas between computability and complexity theory and, thus, for a potential economy of presentation which may be beneficial to the further development of the subject.

References


Appendix: Iterating step gives a PCA

We first verify that $k \cdot x \cdot y = x$ (where $\cdot$, as usual, associates to the left). Here $\text{step}(k, x, []) = \sigma_0(\text{end}, k(x), [])$ has $\text{step}(\text{end}, k_0(x), []) = \sigma_1(k_0(x))$ so that $(k \cdot x) = k_0(x)$. But now

$$k_0(x) \cdot y = \text{step}^\dagger(k_0(x), y, [])$$

$$= \begin{cases} 
\sigma_0(c, v, d) \mapsto \text{step}^\dagger(c, v, d) \\
\sigma_1(x) \mapsto x 
\end{cases} \text{step}(k_0(x), y, [])$$

$$= \begin{cases} 
\sigma_0(c, v, d) \mapsto \text{step}^\dagger(c, v, d) \\
\sigma_1(x) \mapsto x 
\end{cases} \sigma_1(x)$$

$$= x$$

which verifies that $(k \cdot x) \cdot y = x$.

Next, we need $s \cdot x \cdot y$ to be as defined as $x$ and $y$. But clearly $s \cdot x = s_0(x)$ and $s_0(x) \cdot y = s_1(x, y)$ so, as $s_1$ is total this requirement of the combinator is met.

Next, we calculate

$$((s \cdot x) \cdot y) \cdot z = s_1(x, y) \cdot z$$

$$= \text{step}^\dagger(s_1(x, y), z, [])$$

$$= \begin{cases} 
\sigma_0(c, v, d) \mapsto \text{step}^\dagger(c, v, d) \\
\sigma_1(x) \mapsto x 
\end{cases} \text{step}(s_1(x, y), z, [])$$

$$= \text{step}^\dagger(x, z, \text{cons}(c_1(y, z), []))$$

$$= \text{step}^\dagger(\text{end}, x \cdot z, \text{cons}(c_1(y, z), []))$$

$$= \text{step}^\dagger(y, z, \text{cons}(c_0(x \cdot z), []))$$

$$= \text{step}^\dagger(\text{end}, y \cdot z, \text{cons}(c_0(x \cdot z), []))$$

$$= \text{step}^\dagger(x \cdot z, y \cdot z, [])$$

$$= (x \cdot z) \cdot (y \cdot z)$$

Here we use repeatedly the identity:

$$\text{step}^\dagger(x, y, S) = \text{step}^\dagger(\text{end}, \text{step}^\dagger(x, y, []), S) = \text{step}^\dagger(\text{end}, x \cdot y, S)$$

which is true by virtue of the fact that the trace is defined inductively. More precisely, we have

$$\text{step}^{(n)}(x, y, s) = \bigcup_{i+j=n+1} \text{step}^{(i)}(\text{end}, \text{step}^{(j)}(x, y, []), s).$$

The left hand expression is empty unless the iteration terminates in that number of steps. If there is a (first) stage $j$ at which the left hand iteration returns the stack to its original state and the first coordinate is end then $j + 1$ can be used to terminate the inner loop on the right hand side to bring the two iterations to the
same state. Subsequently the result will be the same. If there is no such \( j \) both sides will be the empty map.
Abstract Local Reasoning for Concurrent Libraries: Mind the Gap

Philippa Gardner, Azalea Raad, Mark Wheelhouse and Adam Wright

Department of Computing, Imperial College London, UK

Abstract

We study abstract local reasoning for concurrent libraries. There are two main approaches: provide a specification of a library by abstracting from concrete reasoning about an implementation; or provide a direct abstract library specification, justified by refining to an implementation. Both approaches have a significant gap in their reasoning, due to a mismatch between the abstract connectivity of the abstract data structures and the concrete connectivity of the concrete heap representations. We demonstrate this gap using structural separation logic (SSL) for specifying a concurrent tree library and concurrent abstract predicates (CAP) for reasoning about a concrete tree implementation. The gap between the abstract and concrete connectivity emerges as a mismatch between the SSL tree predicates and CAP heap predicates. This gap is closed by an interface function $I$ which links the abstract and concrete connectivity. In the accompanying technical report, we generalise our SSL reasoning and results to arbitrary concurrent data libraries.

Keywords: Concurrency, Abstraction, Refinement, Separation, Translation, Reasoning

1 Introduction

Local reasoning was first introduced in separation logic to reason about the RAM memory model. Since then, there has been considerable work on combining local reasoning with abstraction. There are two main approaches. One approach starts with concrete reasoning about code that manipulates the standard heap and then builds up layers of abstraction: this approach is used by concurrent abstract predicates (CAP) and its variants [1,16,17], and is ideal for reasoning about the concurrent library java.util.concurrent where the library functions are built up using implementations. The other approach provides direct abstract specifications of abstract code manipulating abstract models, and then justifies the specification by refining it to a correct concrete implementation: this approach is used by context logic [7,9], and is ideal for reasoning about libraries such as POSIX and sequential DOM where the library specification is not grounded on implementation.

1 Email: {pg, azalea, mjw03, adw07}@doc.ic.ac.uk
These current abstraction and refinement approaches have a significant gap in their reasoning. With abstraction, the implementation details leak into the abstraction. For example, consider a CAP predicate $\text{tree}(t)(i,j)(l,u,r)$ for describing tree fragments. The predicate is parameterised by an abstract tree $t$, with concrete pointers $(i,j)(l,u,r)$ describing the concrete interface used to connect the concrete tree fragments. In this case, the concrete interface consists of the first $(i)$ and last $(j)$ nodes of the tree fragment, and the parent $(u)$, left $(l)$ and right $(r)$ siblings. A different implementation, say one using lists, would require a different concrete interface. Thus, this tree predicate is not abstract enough to reason abstractly about updating tree fragments. The missing piece is an abstract way of splitting and combining tree fragments, and a way of linking it to the concrete interface given by $(i,j)(l,u,r)$. With the refinement approach, we have examples where the specification is truly abstract (e.g. [12,19]), but they are typically justified by a soundness result to an operational semantics rather than an implementation. For example, truly abstract reasoning of a tree module (such as DOM) works with predicates based on connecting tree fragments using contexts and place holders. In contrast, the concrete reasoning about an implementation uses pointers. We need to bridge the gap between the abstract connectivity of abstract data structures and the concrete connectivity of concrete heap representations.

We introduce \textit{Structural Separation Logic (SSL)} for reasoning about concurrent abstract data libraries [2,19]. SSL is underpinned by a particular general approach to abstract connectivity for structured data. In this paper, we use a simple concurrent tree library to illustrate our ideas. We provide concrete reasoning about a tree implementation of the library using CAP [1]. The gap between the abstract and concrete connectivity emerges as a mismatch between SSL tree predicates and CAP tree predicates. This gap is closed by an interface function $I$ which links the abstract and concrete connectivity. In the accompanying technical report [18] and Raad’s forthcoming thesis, we generalise SSL and our results to arbitrary structured data libraries. The work presented here does depend on the particular SSL approach to abstract connectivity. Our ideas should, however, apply whenever there is a mismatch between abstract and concrete connectivity.

SSL supports reasoning about fine-grained abstract data fragments stored in abstract heaps. Abstract heaps contain cells identified by abstract addresses (e.g. address $x$) whose values are the disjoint data fragments. These data fragments contain context holes, also given by abstract addresses, which are placeholders for the data fragments found at the appropriate abstract cells. For example, the SSL predicate $\text{ATree}(t)(x)$ describes a tree cell with abstract address $x$ containing tree context $t$. We can split (abstractly allocate) this predicate to obtain the semantically-equivalent assertion $\exists y. \text{ATree}(t_1)(x) \ast \text{ATree}(t_2)(y)$ with $t = t_1[t_2/y]$. The assertion represents the same underlying tree fragment, just in two disjoint parts. Reasoning about semantically-equivalent assertions is only possible due to

---

2 The same issue arises with the well-known predicate $\text{listseg}([1,2,3])(i)(r)$ which describes list fragments with concrete address $i$, abstract contents $1,2,3$, and concrete right pointer $r$. This predicate is appropriate for implementations using singly-linked lists, but not for those using doubly-linked lists which require an additional concrete left pointer. The predicate is not abstract enough because the concrete interface is leaking into the predicate. The missing bit is the abstract connectivity of the list fragment.
We provide a natural implementation of our tree library and show that it is correct with respect to our abstract SSL specification, by relating abstract SSL specifications of the library functions with CAP-like specifications of the concrete function implementations. To do this, we must extend CAP predicates with an interface function \( I \) which relates abstract addresses with concrete pointers. For example, consider the abstract tree predicate \( \text{ATree}(t)(x) \) and the corresponding concrete CAP predicate \( \text{CTree}^I(t)(x) \), where the interface function \( I \) relates the abstract address \( x \) with the pointer interface \((i,j)(l,u,r)\). We prove that our implementation is correct with respect to our abstract specification, using a \( \ast \)-preserving translation (analogous to locality-preserving translation in [9]) parameterised by \( I \).

In this paper, we concentrate on refinement. It is trivial to adapt our work to the approach of fictional separation logic (FSL) [13], where translations are incorporated within the Hoare derivation. However, our choice of translation differs fundamentally from FSL. FSL is designed to reason about sequential programs using \( \ast \)-breaking translations (analogous to locality-breaking translation in [9]): the proofs of FSL assume all possible frames are preserved during the execution of program. While this is a reasonable assumption when reasoning about sequential programs, it is non-trivial to establish its soundness for concurrent programs. One possible way to demonstrate its correctness is to provide linearisability proofs to show that all frames are indeed preserved and that the program behaves atomically. In contrast, our \( \ast \)-preserving translation ensures that compatible stable resources at the abstract level \((p \ast q)\) yield compatible stable resources once translated \((p' \ast q')\).

The correctness of concurrent programs then follows immediately from the disjoint concurrency rule of concurrent separation logic [20].

With abstraction using CAP, we start by concretely reasoning about the heap and then build up levels of abstraction in such a way that \( \ast \) is preserved. As demonstrated, current CAP reasoning leaks implementation details into the abstractions. We believe this can be rectified by extending the abstraction rule to hide the interface functions as well as the predicate interpretations.

We believe that interface functions \( I \) have been barely studied in the literature. They do appear in [9] for reasoning about sequential libraries using context logic. Context logic is not fine-grained enough to extend to concurrency, since it uses a non-commutative separating application unsuitable for use with the disjoint concurrency rule. SSL reasoning makes this extension possible. SSL is used in [19] for specifying sequential POSIX, with the aim to extend to concurrent POSIX in future. However, soundness was proved by comparison with an abstract operational semantics, so the relationship between abstract and concrete connectivity did not arise. In [15,16,13], there is an emphasis on interpretations which relate abstract and concrete states. However, there is no mention of a mapping \( I \) between abstract and concrete interfaces since the connectivity in the examples studied is simple.

---

\(^3\) Note that \( p \ast q \) does not necessarily suggest physically disjoint resources; rather two compatible resources that can be composed together providing a fiction of disjointness.
2 Structural Separation Logic: Tree Library

Structural separation logic (SSL) is a general program logic for specifying structured data libraries and reasoning locally about client programs which call such libraries. Here, we give the intuition and technical details of SSL using an abstract tree library. We give the general theory of SSL in the accompanying technical report [18]. Further details, including a wide number of examples, can be found in [2].

2.1 Intuition

We give our axiomatic SSL specification of a simple deleteTree(n) command. Intuitively, this command removes the entire subtree whose top node identifier corresponds to the value of variable n, leaving the rest of the tree unchanged. We formalise this English description using assertions which describe abstract heaps.

Abstract heaps store abstract data fragments. For instance, Fig. 1a illustrates an abstract heap describing a variable cell n with value n, and a tree cell R with a complete abstract tree as its value. This tree consists of a subtree n[l] with parent u, and left and right siblings l and r with no children. It abstracts away from how a tree might be concretely represented in a machine.

Intuitively, the deleteTree(n) command only affects the subtree identified by n. Abstract heaps enable structured data to be split to provide direct access to this subtree by imposing additional instrumentation using abstract addresses. Consider the transition from Fig. 1a to 1b. Fig. 1a contains an abstract heap with a complete tree at R. We split this complete tree using abstract allocation to obtain the abstract heap in Fig. 1b with subtree n[l] at a fresh, fictional abstract cell x and an incomplete tree at R with a context hole x indicating the position to which the subtree will return. The subtree at n can now be accessed directly. Once the updates have been achieved, the heap can be joined back together using abstract deallocation, as in the transition from Fig. 1b to 1a.

The axiomatic specification of deleteTree(n) (Fig. 1c) is formalised as:

\{\text{var}(n,n) \times \text{ATree}(n[\text{isComplete}])\}(x)\} \text{deleteTree(n)} \{\text{var}(n,n) \times \text{ATree}(\emptyset)(x)\}

The precondition describes a variable store, in which variable n has value n and abstract cell x has complete subtree with top node n as its value\(^4\). Since the subtree

\(^4\) Note that the precondition is a pair consisting of a variable assertion and a subtree (heap) assertion. We use the variables-as-resource model [14]. However, in contrast to the assertions of variables-as-resource where heap and variable assertions are combined using +, we keep the two components separate. As we
at \(x\) is complete, we own the exclusive right to update its contents. Similarly, the postcondition states that the result of the update is an empty tree at abstract cell \(x\), while variable \(n\) remains unchanged. Note that the abstract cell \(x\) must be preserved in order to join this tree fragment with the tree it was split from, using abstract deallocation. The footprint of this command is small in the sense that it intuitively captures the minimum resources required for safe execution of deleteTree(n).

With this axiomatisation, we can verify that the simple client program deleteTree(1) || deleteTree(n) deletes two disjoint subtrees concurrently. Consider the proof derivation in Fig. 2 with program variables 1 and \(n\), their values \(l\) and \(n\), and an abstract tree predicate \(\text{ATree}(u[l \otimes n[t] \otimes r])(R)\) denoting a complete tree at \(R\) containing a subtree with top node \(n\), left and right siblings with top nodes \(l\) and \(r\), and parent \(u\). The initial precondition describes the complete tree at \(R\). To get at the two subtrees, we split the tree twice using abstract allocation, placing the subtrees at nodes \(l\) and \(n\) at the freshly allocated cell addresses \(y\) and \(x\), respectively. We apply the standard separation logic rules of existential elimination and frame to temporarily set aside the partial tree at \(R\) as it is not being updated by the program. The resulting state can then be split into two disjoint parts using the disjoint concurrency rule. Both parts are now in the right form to match the precondition of the deleteTree axiom. The two updates can happen resulting in the postconditions with empty trees at \(y\) and \(x\) which are joined together by the disjoint concurrency rule. We reintroduce the state set aside through the existential elimination and frame rule, and obtain the complete tree at \(R\) using abstract deallocation twice to remove \(y\) and \(x\).

---

demonstrate in §4, this is because when translating a library, only heaps are transformed by the translation while program variables are simply preserved.
2.2 Technical Details

We give the technical details of SSL using the tree library $T$ discussed in §2.1.

**Definition 2.1** The set of abstract atomic tree commands of $T$ are defined as:

$$\text{ATOM}_T := n := \text{getFirst}(n) \mid m := \text{getRight}(n) \mid m := \text{getUp}(n)$$

$$\mid m := \text{newNodeAfter}(n) \mid \text{deleteTree}(n) \mid \text{appendChild}(m, n)$$

for all program variables $n, m \in \text{PVars}$. Our commands are chosen to demonstrate a wide range of structural manipulations. The command $\text{getFirst}(n)$ returns the identifier of the first child of node $n$, or null if $n$ has no children. The $\text{getRight}(n)$ and $\text{getUp}(n)$ commands behave analogously with respect to the right sibling and parent of node $n$, respectively. The command $\text{newNodeAfter}(n)$ creates a new node with a fresh identifier, making it the right sibling of node $n$ and returning the fresh identifier. The $\text{deleteTree}(n)$ command removes the entire subtree identified by $n$ from the tree. Finally, the command $\text{appendChild}(m, n)$ moves the subtree at $n$ to be the last child of node $m$. Each of these commands faults if any of the nodes given as parameters are not present in the tree. Additionally, the $\text{appendChild}(m, n)$ command faults if $m$ is a descendant of $n$. These commands are intended to be used with any programming language. In this paper, we use the programming language of the Views framework [6] instantiated with $\text{ATOM}_T$ as the set of atomic commands. We write $\text{PROG}_T$ to denote the set of programs written in this language.

We specify the tree commands using abstract heaps with cells whose addresses are either abstract addresses or the tree root address $R$, and whose values are abstract trees.

**Definition 2.2** Given a countably infinite set of abstract addresses $\text{AAdd}$ ranged over by $x, y, z$, the set of addresses for the abstract tree heaps is $\text{Add}_T = \{ R \} \cup \text{AAdd}$.

We now define abstract trees, which can either be complete trees or tree fragments with abstract addresses for context holes.

**Definition 2.3** Given the set of abstract addresses $\text{AAdd}$, the set of abstract trees $\text{Data}_T$, ranged over by $t, t_1, \ldots, t_n$, is defined inductively as:

$$t := \emptyset \mid n[t] \mid t_1 \otimes t_2 \mid x$$

where $n \in \mathbb{N}^+, x \in \text{AAdd}$, the sets $\text{AAdd}$ and $\mathbb{N}^+$ are disjoint, and no tree contains duplicate node identifiers or abstract addresses. Abstract trees are equal up to the associativity of $\otimes$ with unit $\emptyset$. For brevity, we write $n$ for $n[\emptyset]$. There is an associated identifiers function $\text{ids} : \text{Data}_T \rightarrow \mathcal{P}(\mathbb{N}^*)$ and an associated addresses function $\text{addr} : \text{Data}_T \rightarrow \mathcal{P}(\text{AAdd})$ which respectively, return the sets of node identifiers and abstract addresses present in an abstract tree. The application $t_1 \otimes x t_2$ is standard: it is undefined when $x \notin \text{addr}(t_1)$ and is otherwise defined as $t_1[t_2/x]$, denoting the standard substitution of $t_2$ for $x$ in $t_1$.

An abstract tree heap is a mapping from addresses to abstract trees.

---

5 Strictly speaking, this grammar describes forests or ordered lists of abstract trees. We use the term forest when we wish to emphasise the list with a first and last element.
Definition 2.4 The set of abstract tree heaps $H_T : \text{ADD}_T \xrightarrow{\text{fin}} \text{DATA}_T$, ranged over by $h, h_1, \ldots, h_n$, is the set of functions from addresses to abstract trees such that for all $h \in H_T$ the following restrictions hold:

\[ \forall a_1, a_2 \in \text{ADD}_T. \quad a_1 = a_2 \lor \text{addrs}(h(a_1)) \cap \text{addrs}(h(a_2)) = \emptyset \]

\[ \forall a_1, a_2 \in \text{ADD}_T. \quad a_1 = a_2 \lor \text{ids}(h(a_1)) \cap \text{ids}(h(a_2)) = \emptyset \]

where the descendent relation $D$ for heap $h$ is defined as:

\[ a \overset{D}{\rightarrow} h y \iff y \in \text{addrs}(h(a)) \]

and $D^+_h$ denotes its transitive closure.

The first two restrictions state that the abstract addresses being used as context holes and the node identifiers are unique across an abstract heap. The last condition states that the abstract addresses join up to produce sensible abstract trees. For instance, $\{(x \rightarrow m[y], y \rightarrow n[x])\}$ is not an abstract tree heap because of the cycle. An abstract tree heap may be: complete, with no use of abstract addresses; complete, but with the tree split across several heap cells; or incomplete, missing some heap cells needed to join some abstract addresses. Incomplete abstract heaps are necessary for local reasoning using the frame rule. In this case, there will be some choice for the missing cells that would render the tree complete.

Given an abstract tree heap $h$, $h^{\text{in}}$ and $h^{\text{out}}$ denote the set of abstract cell addresses and context holes, respectively:

\[ h^{\text{in}} \triangleq \text{ADD} \cap \text{dom}(h) \quad h^{\text{out}} \triangleq \text{ADD} \cap (\bigcup_{t \in \text{co-dom}(h)} \text{addrs}(t)). \]

By design, abstract tree heaps are similar to standard heaps. The construction of a separation algebra over them is thus straightforward.

Definition 2.5 The separation algebra of abstract tree heaps is $A_T \triangleq (H_T, \bullet_T, 0_T)$, where $H_T$ is given in Def. 2.4, $\bullet_T$ is the standard disjoint function union with the proviso that the resulting abstract heap is well-formed as per Def. 2.4; and $0_T$ is a singleton set consisting of the partial function with an empty domain and co-domain.

Since we use variables as resource [14], we pair variable stores for declaring the values of variables with our abstract tree heaps.

Definition 2.6 The separation algebra of abstract tree states $\mathcal{S}A_T \triangleq (\Sigma \times H_T, \star_{\Sigma} \times \star_T, 0_{\Sigma} \times 0_T)$, is the Cartesian product of the separation algebra of variables $(\Sigma, \star_{\Sigma}, 0_{\Sigma})$ [14] and the algebra of abstract tree heaps (Def. 2.5).

To reason about our tree programs, we use the program logic of the Views framework as described in [6]. We instantiate the framework with the separation algebra of abstract tree states (Def. 2.6) as the view monoid and the tree library commands (Def. 2.1) as the atomic commands. What remains is to describe the axioms associated with the tree library commands (Def. 2.7).

Our views are sets of abstract tree states in $\mathcal{P}(\Sigma \times H_T)$. To increase readability of variable sets, we write $\text{var}(x, v)$ for $\{x \rightarrow v\}$ and $p \star q$ for $\{\sigma_1 \star_{\sigma} \sigma_2 | \sigma_1 \in p \land \sigma_2 \in q\}$.
Transitions are labelled either by atomic commands or by id which labels computation steps in which states are not changed. We extend the behaviour of the id transitions of Views by declaring the relation \text{AXIOM}_\text{id} for abstract allocation/deallocation. Abstract (de)allocation does not change the underlying program states and can therefore be seen as id transitions.

**Definition 2.8** The identity axiomatisation: \text{AXIOM}_\text{id} : (\Sigma \times H_\Sigma) \times \{\text{id}\} \times (\Sigma \times H_\Sigma)
is given by the abstract allocation and deallocation axioms:

\[
\{ \text{ATree}(d_1 \circ_X d_2)(a) \} \ 	ext{id} \ \{ \exists y. \ \text{ATree}(d_1 \circ_X y)(a) \} \ast \text{ATree}(d_2)(y) \}
\]

\[
\{ \exists y. \ \text{ATree}(d_1 \circ_X y)(a) \} \ast \text{ATree}(d_2)(y) \} \ 	ext{id} \ \{ \text{ATree}(d_1 \circ_X d_2)(a) \}
\]

The existential quantification of the abstract address \( y \) is analogous to the existential quantification used for heap allocation in separation logic.

**Definition 2.9** Given the set of abstract tree heaps \( H_T \) (Def. 2.4), the set of atomic commands \( \text{ATOM}_T \) (Def. 2.1) and their axiomatisation \( \text{AXIOM}_T \) (Def. 2.7), the abstract tree library \( T \) is defined as: \( T = (H_T, \text{ATOM}_T, \text{AXIOM}_T \cup \text{AXIOM}_{Id}) \)

**Definition 2.10** Given abstract tree states \( p, q \in \text{P} (\Sigma \times H_T) \) (Def. 2.6) and program \( C \in \text{PROG}_T \) (Def. 2.1), an abstract triple is \( \Omega \in T \{ p \} C \{ q \} \). \( \Omega \in \text{PEnv}_T \) is a procedure specification environment which is a set of procedure specifications \( f : p_1 \Rightarrow q_1 \), where \( f \) is a procedure name and \( p_1, q_1 \in \text{P} (\Sigma \times H_T) \) are pre/post-conditions of \( f \).

## 3 Concurrent abstract predicates: Tree Implementation

We use concurrent abstract predicates (CAP) to reason about our implementation. We describe the concrete representation of our abstract trees and the concrete implementation of the tree commands in §3.1. We reason about our implementation in §3.2. We give an informal account of how to establish its correctness with respect to the abstract specification in §3.3, and give the formal justification in §4.

### 3.1 Concrete Tree Implementation

**Tree Representation** We give a concrete representation of the abstract tree heaps introduced in §2. Consider the abstract tree heap depicted in Fig. 1a. The corresponding concrete tree heap is illustrated in Fig. 4. Each tree node is represented by a node cell with pointers to its left sibling, parent, first child, last child and right sibling where \( \rightarrow \) denotes a null pointer. With the abstract tree heap, the deletion of the subtree rooted at \( n \) is a self-contained operation; it does not rely on the context surrounding the subtree. However, this is not the case for the implementation. Since each node cell maintains pointers to its siblings, deletion of the subtree at \( n \) requires altering the outgoing pointers of its siblings (and sometimes the parent). In this example, the right pointer of node cell \( l \) must be redirected to \( r \) and vice-versa; only then can the subtree at \( n \) be discarded. Furthermore, in the implementation of the concurrent client program \texttt{deleteTree(l) || deleteTree(n)}, both executing threads need to access the pointers between \( l \) and \( n \) simultaneously, which calls for a suitable synchronisation technique. In our implementation, we synchronise access to these pointers through locking. Each of the left, first, last and right pointers are protected.
by corresponding locks \((lL, dL, eL, rL)\) that are to be acquired by the competing threads beforehand. We refer to these locks as pointer locks.

We represent each tree node as a node cell consisting of a block of nine consecutive heap cells, \(n \rightarrow l, u, d, e, r, lL, dL, eL, rL\) where the first five cells contain pointers to the siblings, parent and children, and the last four cells contain the pointer locks. The primary reason for choosing this particular representation was its resemblance to the representation of tree nodes in the Document Object Model (DOM) implementation of the WebKit project\(^6\). To increase readability, we write \(n.l, n.u, \ldots, n.rL\) for \(n, n+1, \ldots, n+8\), respectively.

**Tree Implementation** We provide a concrete implementation of each of the abstract tree commands. Fig. 5 shows our implementation of the deleteTree command with some auxiliary procedures. The implementation needs to cater for various cases regarding the siblings of node \(n\) such as when \(n\) does not have a left or right sibling. The implementation of the remaining commands are given in the technical report [18]. Note that this implementation is prone to deadlocks; this can occur with our example program deleteTree(1) || deleteTree(n) when 1 and \(n\) are adjacent siblings. We focus on this simple implementation here as it is sufficient to describe our ideas. We reason about a deadlock-free implementation in [18].

---

6 However, WebKit supports only sequential DOM manipulation and does not use locks.
3.2 Reasoning about Concrete Tree Implementation

Recall that the pointers between two sibling nodes (or sometimes a parent and child node) are \textit{shared} resources accessible by both nodes. We use concurrent abstract predicates (CAP) \cite{1} to manage this sharing.

\textbf{Concurrent Abstract Predicates} With CAP, the state is modelled as a pair consisting of a thread-local state and a shared state. The shared state is divided into a set of regions, each encompassing some shared portion of the state. Each region is identified by a region identifier $R$ and is governed by a protocol that describes how the resources of the region can be manipulated. For instance, a lock resource at location $x$ can be specified by:

\[
\text{lock}(x) \triangleq \exists R, \pi. [\mathcal{L}]_R^R \ast \begin{cases} \text{Unlocked}(R, x) \lor \text{Locked}(R, x) \\ \text{T}(R, x) \end{cases}
\]

where $\text{Unlocked}(R, x) \triangleq x \mapsto 0 \ast [\mathcal{U}]_1^R$ and $\text{Locked}(R, x) \triangleq x \mapsto 1$. This lock definition states that there exists a shared region $R$ containing the lock at location $x$ and the thread’s local state contains some (for permission $\pi \in (0, 1]$) locking capability $[\mathcal{L}]_\pi^R$ to acquire it. The region is in one of two states: either the lock is unlocked ($x \mapsto 0$) and the region holds the full capability $[\mathcal{U}]_1^R$ to unlock it; or the lock is taken ($x \mapsto 1$) and the unlocking capability has been claimed by the locking thread.

The protocol of a shared region is specified through a set of actions, such as actions $\mathcal{L}$ and $\mathcal{U}$ for the lock example. A thread can perform an action if it has a non-zero capability for that action (such as the capability $[\mathcal{L}]_\pi^R$ for $\pi > 0$). The actions permitted on the lock region are declared in $\mathcal{T}(R, x)$:

\[
\mathcal{T}(R, x) \triangleq \begin{cases} \mathcal{L} : (x \mapsto 0 \ast \mathcal{U}_1) & \rightarrow x \mapsto 1 \\ \mathcal{U} : x \mapsto 1 & \rightarrow (x \mapsto 0 \ast \mathcal{U}_1) \end{cases}
\]

The action associated with $\mathcal{L}$ changes the state of the shared region from unlocked to locked, moving the capability $[\mathcal{U}]_1^R$ to the thread’s local state. The action associated with $\mathcal{U}$ behaves dually, returning $[\mathcal{U}]_1^R$ from the local state to the shared region.

The definition of the CAP separation algebra is given in \cite{1,6}. It provides a set of instrumented states $M_H$ consisting of a local state, a shared state and an action relation capturing the ways in which the shared state can be manipulated. The definition is parametrised by an underlying separation algebra for describing the local state. For this paper, we work with the CAP separation algebra instantiated with the standard fractional heap separation algebra.

\textbf{Definition 3.1} The \textit{CAP separation algebra}, instantiated with fractional heap separation algebra, is $A_{CH} \triangleq (M_H \cdot \cdot C_H, 0_C_H)$. The \textit{separation algebra of CAP states} is $SA_{CH} \triangleq (\Sigma \times M_H \cdot \cdot \cdot C_H, 0_H \cdot \cdot \cdot C_H)$, where $\times$ denotes standard Cartesian product.

Recall that we base our reasoning on the Views framework \cite{6} which provides general reasoning about a generic programming language parametrised by a set of atomic commands and their axiomatisation.

\textbf{Definition 3.2} Let $\text{ATOM}_H$ be the set of atomic heap commands including assignment, value lookup, memory allocation/deallocation and the compare and set
construct CAS. We write \( \text{PROG}_H \) for the set of programs generated from Views’ programming language instantiated with set \( \text{ATOM}_H \). Let \( \text{AXIOM}_H \) be the standard set of separation-logic axiomatisations for these commands based on the fractional heap separation algebra. The Views framework provides the associated reasoning for \( \text{PROG}_H \).

**Definition 3.3** Given the CAP separation algebra \( \mathcal{A}_C_H \) (Def. 3.1), the atomic heap commands \( \text{ATOM}_H \) and their axiomatisation \( \text{AXIOM}_H \), the CAP library \( C_H \) for fractional heaps is defined as: \( C_H \equiv (\mathcal{A}_C_H, \text{ATOM}_H, \text{AXIOM}_H) \).

**Definition 3.4** Given CAP states \( p, q \in \mathcal{P}(\Sigma \times M_H) \) (Def. 3.1) and program \( C \in \text{PROG}_H \) (Def. 3.2), a concrete triple is \( \Omega \equiv \langle p \Rightarrow q \rangle \), where \( \Omega \in \text{PEnv}_{C_H} \) is a set of procedure specifications \( f : p_1 \Rightarrow q_1 \), where \( f \) is a procedure name and \( p_1, q_1 \in \mathcal{P}(\Sigma \times M_H) \) are pre/post-conditions of \( f \).

In §4, we define the library refinement \( \tau : T \rightarrow C_H \) for abstract tree library \( T \) (Def. 2.9) and concrete CAP library \( C_H \) (Def. 3.3). As a first step in defining \( \tau \), we identify the concrete CAP states corresponding to the abstract tree fragments. These concrete CAP states rely on an interface function \( I \) linking abstract addresses with concrete pointers.

**Interface Functions.** Fig. 4 illustrates a concrete heap representation of a complete abstract tree at root address \( R \). However, many of the abstract tree commands (e.g. `deleteTree`) are specified using tree fragments rather than complete trees. We need to understand the concrete representation of tree fragments.

When an abstract tree heap is split using abstract allocation, the constituent heaps are agnostic to one-another’s shapes. For instance, consider the abstract tree heap \( R \rightarrow l \otimes i \otimes j \otimes r \) which can be split as \( h_1 \bullet h_2 \) where \( h_1 \equiv R \rightarrow l \otimes x \otimes r \) and \( h_2 \equiv x \rightarrow i \otimes j \). The abstract tree heap \( h_1 \) embodies no knowledge of the tree placed within \( x \); \textit{mutatis mutandis} for \( h_2 \). At the concrete level, the situation is different. The concrete representation of \( h_2 \) relies on additional knowledge from the concrete representation of \( h_1 \), since the representation of node \( i \) includes pointers to its left sibling (\( l \)) and parent (\( u \)). Thus, when translating an abstract heap with abstract addresses, we require supplementary information originating from the concrete heap. We track this additional piece of information associated with each abstract address \( x \in \text{AADD} \) through a concrete \textit{interface}. Consider Fig. 6, which depicts an abstract tree at abstract cell \( x \) (left) and its concrete representation (right) assuming an interface function \( I \).

In the concrete representation, the solid lines represent resources held locally, such as node \( n \), its up pointer and the subtree \( t \), while the dashed lines denote shared resources, such as the pointers between node \( n \) and its siblings. A concrete \textit{in-interface} records the address of the first (\( i \)) and last (\( j \)) nodes of the concrete representation of the forest pointed to by \( x \); in our example, the in-interface is \( in \equiv (n,n) \). A concrete \textit{out-interface}
ICTree\(_r\) \((R)\) \(\triangleq \exists i,j.\) ICTree\(_r\) \((t)\) \(\left(\text{null, null, null}\right)\)

\[
\text{ICTree}_r(t)(x) \triangleq \text{ICTree}_r(t)(i^x, j^x)(l^x, u^x, r^x)
\]

\[
\ast I_r^\text{in}(x) \triangleq (i^x, j^x) \ast I_r^\text{out}(x) \triangleq (l^x, u^x, r^x)
\]

\[
\text{ICTree}_r(\emptyset)(i,j)(l,u,r) \triangleq (i \Leftarrow r) \ast (j \Leftarrow l)
\]

\[
\text{ICTree}_r(x)(i,j)(l,u,r) \triangleq I_r^\text{in}(x) \triangleq (i,j) \ast I_r^\text{out}(x) \triangleq (l,u,r)
\]

\[
\text{ICTree}_r(t_1 \otimes t_2)(i,j)(l,u,r) \triangleq \exists p,q.\) ICTree\(_r\) \((t_1)\) \((i,p)(l,u,q)\)
\]

\[
\ast \text{ICTree}_r(t_2)(q,j)(p,u,r)
\]

\[
\text{ICTree}_r(n[i])(i,j)(l,u,r) \triangleq i \Leftarrow j \Leftarrow n \ast n.u \rightarrow u \ast \exists d,e.
\]

\[
\left(\begin{array}{c}
\text{Left}(n, l, u) \ast \text{First}(n, d) \ast \text{Last}(n, e) \ast \text{Right}(n, r, u)
\end{array}\right)
\]

\[
\ast \text{ICTree}_r(t)(d,e)(\text{null, null, null})
\]

Fig. 7. CAP Representation of Abstract Tree Heaps: we write \(a \Leftarrow b\) for \(a = b \land \text{emp.}\)

records the addresses of the parent node \((u)\) and nodes placed immediately to the
left \((l)\) and right \((r)\) of the abstract address \(x\); in our example, the out-interface is
\(\text{out} \triangleq (l^x, u^x, r^x)\). The interface function \(I_r\) maps \(x\) to \((\text{in}, \text{out})\).

**Definition 3.5** The sets of concrete in- and out-interfaces are defined by:

\[
\text{IN}_r \triangleq (N^+ \cup \{\text{null}\})^2 \quad \quad \text{OUT}_r \triangleq (N^+ \cup \{\text{null}\})^3
\]

The set of in- and out-interface functions are defined by \(I_r^\text{in} \triangleq \mathcal{P}(AADD \rightarrow \text{IN}_r)\) and
\(I_r^\text{out} \triangleq \mathcal{P}(AADD \rightarrow \text{OUT}_r)\). The set of interface functions is \(I_r \triangleq I_r^\text{in} \times I_r^\text{out}\).

**CAP Representation of Trees** Fig. 7 defines concurrent abstract predicates for
describing the concrete representation of abstract tree heaps, parameterised by
interface function \(I_r\). The CTree\(_r\) \((t)(R)\) predicate provides a concrete representation
of the tree fragment \(t\) at root address \(R\). The predicate CTree\(_r\) \((t)(x)\) provides a
concrete representation of the tree fragment at abstract address \(x\). Both predicates
are defined using a ICTree predicate indexed by in- and out-interfaces determined
by \(I_r\). For \(R\), the out-interface is \((\text{null, null, null})\). The in-interface is unknown at
this point and hence is existentially quantified. Its value is later found using the
ICTree predicate. The interface of \(x\) is determined by the interface function \(I_r\).

We now describe the ICTree predicate with the explicit in- and out-interface
arguments \((i,j)\) and \((l,u,r)\), following the cases of the inductive definition.

**ICTree\(_r\) (\(\emptyset\))** There is no resource in the concrete heap representation, simply some
pointer equalities. The assertion \(i \Leftarrow r\) simply states that the address of the first node
in the forest is equal to the address of the node cell immediately to the right of
the tree; similarly for \(j\) and \(l\). This is to ensure the correct concrete representation
of trees such as \(t_1 \otimes \emptyset \otimes t_2\).

**ICTree\(_r\) (\(x\))** There is no resource in the concrete heap representation, simply the
appropriate connection between abstract address $x$ and the interfaces given by $I_r$.

$\text{ICTree}^I_r((t_1 \otimes t_2))$ This is simply the $\ast$-composition of the concrete representations of the constituent abstract subtrees, given an appropriate choice of interfaces.

$\text{ICTree}^I_r(n[t])$ The first two lines provide the concrete representation of the node $n$, and the last line represents the subtree $t$. For the subtree, the concrete representation is straightforward. For the node, $i \equiv j \equiv n$ since there is only one tree in the forest. The concrete representation has full ownership of the parent pointer $n.u \rightarrow u$, as there is no competition for this resource from other nodes. The $\text{Left}$, $\text{Right}$, $\text{First}$ and $\text{Last}$ predicates represent the pointers and locks associated with the left sibling, right sibling, first child and last child, which are in competition with the representations of nearby nodes and hence must be shared.

We give the definition of $\text{Left}$ predicate here; the $\text{Right}$, $\text{First}$ and $\text{Last}$ predicates are defined analogously and we defer them to the accompanying technical report [18].

The definition of the $\text{Left}$ predicate is given in Fig. 8 and is described by the $\text{isLLock}$ and $\text{ownsR}$ predicates.

$\text{isLLock}(n, l, u, 0.5)$ This predicate states that there exists a shared region $R_{nl}$ containing the left pointer lock of node $n$; the thread’s local state contains some locking capability $[\mathcal{L}]_{L, 0.5}^{R_{nl}}$ associated with the region.

$\text{ownsR}(n, l, u, 1)$ Similar to $\text{isLLock}$, this predicate refers to the $R_{nl}$ region where the thread’s local state holds full permission on the $\text{witness}$ capability $[\mathcal{W}]_{L}^{R_{nl}}$. This is to denote that even though the pointers between $n$ and its left sibling $l$ are shared, node $n$ itself is owned by the current thread. Later in the description of LWit predicate we demonstrate how this capability can be used to track the identity of the thread that has claimed the left pointer lock of $n$.

The contents of the $R_{nl}$ region are analogous to that of the lock example given before. The difference is the additional pointer resource ($\otimes$) contained within the region. The region can be in one of two states: $\text{LUnlocked}$ or $\text{LLocked}$.

$\text{LUnlocked}(R_{nl}, n, l, u)$ In this state, the left pointer lock is not taken ($n.lL \rightarrow 0$), and the full unlocking capability ($[\mathcal{U}]_{1}^{R_{nl}}$) and the $\otimes$ resource are in the region.

$\text{LUnlocked}(n, l, u, 0)$ This predicate describes the pointer resources between node $n$ and its left hand side depending on its position in the tree. When $n$ has a left sibling ($l \neq \text{null}$), it consists of partial ownership of the pointers between $n$ and $l$, that is, $n.l \overset{0.5}{\rightarrow} l \ast l.r \overset{0.5}{\rightarrow} n$. It also contains the capability to acquire the right pointer lock of $l$, in order to obtain full ownership of the pointers between node $n$ and $l$. This is captured by the $\text{isRLock}(l, n, u, 0.5)$ predicate. On the other hand, if $n$ does not have a left sibling and is thus the first child of node $u$ ($l = \text{null} \land u \neq \text{null}$), the $\otimes$ resource consists of partial ownership on the left pointer of $n$ and the first pointer of $u$ ($n.l \overset{0.5}{\rightarrow} l \ast u.d \overset{0.5}{\rightarrow} n$) as well as the capability to acquire the first pointer lock of $u$ ($\text{isDLock}(u, n, 0.5)$). Finally, if $n$ is the first child underneath the root address $\mathcal{R}$, and thus both its left sibling and parent correspond to null ($l = u = \text{null}$), the $\otimes$ resource consists of full permission on $n$’s left pointer ($n.l \overset{1}{\rightarrow} l$) and the remaining
locking capability on region $R_{nl}$ ($\text{isLLock}(n,l,u,0.5)$).

**LLocked($R_{nl}, n,l,u$)** In this state, the left pointer lock is taken ($n.LL \rightarrow 1$), and the capability $[\mathcal{U}]_{1}^{R_{nl}}$ and the pointer resources $\mathcal{U}$ have been claimed by the locking thread in exchange for the LWit resource.

**LWit($l,n,u$)** Since the left pointer lock of $n$ can only be be acquired by the thread in possession of node $n$, or node $l$ (or node $u$ if $n$ is the first child of $u$ and does not have a left sibling), the LWit predicate is used to track the identity of the locking thread. Recall from the definition of the Left predicate that the full witness capability $[\mathcal{W}]_{l}^{R_{nl}}$ is held by the thread that owns node $n$, as described by $\text{ownsR}(n,l,u,1)$. We thus use the witness capabilities to determine the identity of the locking thread. The first disjunct denotes the case where the lock has been claimed by the thread in possession of node $n$ and corresponds to the witness capability on this region ($\text{ownsR}(n,l,u,0.5)$). Analogously, the second disjunct
represents the case where the thread in possession of node \( l \) has acquired the lock, denoted by \((\text{ownsR}(n,l,u,0.5))\). The third disjunct captures the case where \( n \) does not have a left sibling and the lock has been taken by the thread that owns the parent node \( u \) \((\text{ownsD}(u,n,0.5))\).

\textbf{LC}(R_{nl}, n, u)\ The R_{nl} region is governed by the LC(R_{nl}, n, u) protocol describing the ways in which its contents can be manipulated through actions. The action associated with \( L \) changes the state of \( R_{nl} \) from \text{LUnlocked} to \text{LLocked}. Dually, the action of \( U \) changes the state of the region from \text{LLocked} to \text{LUnlocked}. The initial/final contents of the region are realised by the \text{LInit} predicate. The definition of \text{LInit} predicate is nonessential as it bears no relevance in understanding the interactions between the threads and the shared region and is thus omitted here.

Recall that abstract trees can be split and joined using abstract allocation and deallocation. We show that concrete trees can be split (joined) analogously provided that the interface function is extended (reduced) accordingly to capture the interface associated with the freshly allocated (deallocated) abstract address.

\textbf{Theorem 3.6 (Abstract (de)allocation)} For all interface functions \( I_{r} \in I_{\tau} \), addresses \( a \in \text{ADD}_{\tau} \) and trees \( t_{1}, t_{2} \in \text{DATA}_{\tau} \):

\[ CTree^{l_{r}}(t_{1} \circ x t_{2})(a) \equiv \exists y \in \text{ADD}, in \in \text{IN}_{r}, out \in \text{OUT}_{r}. CTree^{l_{r}}(t_{1} \circ x y)(a) \ast CTree^{l_{r}}(t_{2})(y) \]

where \( \text{IN}_{r}, \text{OUT}_{r} \) and \( I_{r} \) are given in Def. 3.5 and \( I'_{r} = I_{r} \circ \{[y \mapsto in], [y \mapsto out]\} \).

\textbf{Proof.} The proof of this theorem is provided in the technical report [18].

\subsection*{3.3 Soundness of Concrete Tree Library C_{H}}

In order to show that our concrete CAP library \( C_{H} \) (Def. 3.3) is sound with respect to the abstract tree library \( T \) (Def. 2.9), we show that everything that can be proved about the abstract library \( T \), can also be proved about the concrete library \( C_{H} \). That is, for every abstract triple \( \Omega_{T} = \{p\}C\{q\} \) (Def. 2.10), we show that there exists a corresponding concrete triple \( \Omega'_{C_{H}} = \{p'\}C'\{q'\} \) (Def. 3.4) where \( C' \) is the implementation of \( C \) and \( p', q' \) denote the concrete representations of \( p, q \), respectively. For instance, we need to show that the implementation of the deleteTree command in Fig.5, satisfies its abstract specification as given in Fig.3.

In §4, we formalise triple transformation (Def. 4.6) and define what it means for the CAP library \( C_{H} \) to be sound with respect to the abstract tree library \( T \) (Def. 4.8). We then prove that \( C_{H} \) is sound with respect to \( T \) (Theorem 4.9). In what follows, we give an informal account of establishing the correctness of the deleteTree implementation and state a pseudo-theorem that almost captures the desired result. We defer the formalisation of the correct theorem to §4 (Theorem 4.7), and the full proof to the accompanying technical report [18].
Pseudo-Theorem 1 Let $[\text{deleteTree}(n)]_\tau$ denote the implementation of deleteTree(n) command in Fig. 5. The following statement almost holds:

$$\forall I_c \in \mathcal{I}_\tau. \left\{ \exists I_d \in \mathcal{I}^\text{del}_\tau. (\text{var}(n, n) \times \text{CTree}^{I_c \circ I_d}(n[t])(x)) \right\}$$

$$[\text{deleteTree}(n)]_\tau$$

$$\left\{ \exists I_d \in \mathcal{I}^\text{del}_\tau. (\text{var}(n, n) \times \text{CTree}^{I_c \circ I_d}(\emptyset)(x)) \right\}$$

with $$\mathcal{I}^\text{del}_\tau \triangleq \left\{ I \in \mathcal{I}_\tau \mid \text{dom}(I^{\text{in}}) = \{ x \} \land \text{dom}(I^{\text{out}}) = \emptyset \right\}.$$  

The pre-condition contains the variable resource $\text{var}(n, n)$ and the concrete tree representation $\text{CTree}(n[t])(x)$ of complete subtree at abstract address $x$. The pre-condition gives the right to control and modify the inner interface $I^\text{del}_\tau$ of address $x$ (in this case, $(n, n)$). This freedom to modify $I^\text{del}_\tau$ is captured by the existential quantification of $I_d$ in both pre- and post-conditions. Note that the control over the outer interface of $x$ as well as the interfaces associated with other abstract addresses lies with the context. We therefore need to be agnostic to the interfaces associated with abstract addresses outside the domain of $I_d$ and show that the implementation is correct for all valid choices of context interfaces $I_c$. This is realised by the universal quantification of $I_c$ outside the Hoare triple. However, since we are reasoning about concurrent programs, during the execution of deleteTree, the interfaces captured by $I_c$ are potentially changing underfoot. Hence, contrary to what the above triple states, the value of $I_c$ in the pre-condition does not necessarily agree with that of $I_c$ in the post-condition. The correct theorem is given in §4.

4 Correct Library Translation

We state what it means to correctly implement our abstract tree library $\mathbb{T}$ (Def. 2.9) with our CAP library $\mathbb{C}_\mathbb{H}$ (Def. 3.3) by defining a library translation $\tau : \mathbb{T} \rightarrow \mathbb{C}_\mathbb{H}$. In §4.1, we give our specific library translation $\tau$ and, in §4.2, we prove its soundness. In the accompanying technical report [18], we generalise this approach to translate arbitrary abstract libraries, and stipulate a set of general properties that when satisfied, will warrant sound translation of any abstract library. These properties are stated for our specific library translation in Theorem 4.9.

4.1 Tree to CAP Translation

Our goal is to define a translation $\tau : \mathbb{T} \rightarrow \mathbb{C}_\mathbb{H}$ in such a way that would allow us to correctly transform abstract tree triples to concrete CAP triples, following the spirit of Pseudo-theorem 1. We give an abstract tree heap translation that maps abstract tree heaps to CAP heaps. To keep the translation concise, we define it using CAP assertions which can then be interpreted as elements of $\mathcal{P}(M_\mathbb{H})$ [1].

Definition 4.1 Given the separation algebra of abstract tree heaps (Def. 2.5) $\mathcal{A}_\mathbb{T} \triangleq (H_\mathbb{T}, \bullet_\mathbb{T}, 0_\mathbb{T})$, the tree interface function $\mathcal{I}_\tau$ (Def. 3.5) and the CAP separation algebra (Def. 3.1) $\mathcal{A}_\mathbb{C}_\mathbb{H} \triangleq (M_\mathbb{H}, \bullet_\mathbb{C}_\mathbb{H}, 0_\mathbb{C}_\mathbb{H})$, the abstract tree heap translation $\langle , \rangle_\tau : \mathcal{H}_\mathbb{T} \rightarrow \mathcal{I}_\tau \rightarrow \mathcal{P}(M_\mathbb{H})$ is a function defined inductively over the structure of abstract tree heaps:
\[
\langle 0 \rangle_{\tau}^{I} = \text{emp} \quad \langle \mathcal{R} \to t \rangle_{\tau}^{I} = \text{CTree}'(t)(\mathcal{R}) \quad \langle x \to t \rangle_{\tau}^{I} = \text{CTree}'(t)(x)
\]

\[
\{ h_{1} \cdot_{T} h_{2} \}_{\tau}^{I} = 3 I_{1}, I_{2}, I = I_{1} \cup I_{2} \wedge \langle h_{1} \rangle_{\tau}^{I_{1}} \cdot \langle h_{2} \rangle_{\tau}^{I_{2}}
\]

where the definition of the CTree predicate is as given in Fig. 7.

We also need to transform programs in \(\text{PROG}_T\) to programs in \(\text{PROG}_{\text{H}}\).

**Definition 4.2** The implementation function \([\cdot]_{\tau} : \text{PROG}_T \rightarrow \text{PROG}_{\text{H}}\), replaces each call to an atomic tree command in \(\text{ATOM}_T\) with a call to a correspondingly named procedure denoting its implementation, while other program constructs remain unchanged. The implementation of \(\text{deleteTree}\) is as given in Fig. 5; the implementation function \(\text{does not change by the environment}\). We define the set of stable interface functions

\[
\text{SI} = \text{set of stable inner-interface functions}
\]

and the set of stable outer-interface functions

\[
\text{SO} = \text{set of stable outer-interface functions}
\]

whereby an abstract address is associated not with a single interface but with a set of interfaces. By associating a set of possible interfaces with an abstract address at any one time, we can account for the potential change of interfaces by the environment.

**Definition 4.3** The translation \(\tau : T \rightarrow C_{\text{H}}\) is a triple comprising the set of interfaces \(\text{IN}_{\tau} \times \text{OUT}_{\tau}\) (Def. 3.5), the abstract tree heap translation \(\{\cdot\}_{\tau}^{(3)}\) (Def. 4.1) and the implementation function \([\cdot]_{\tau}\) (Def. 4.2):

\[
\tau \upmodels (\text{IN}_{\tau} \times \text{OUT}_{\tau}, \{\cdot\}_{\tau}^{(3)}, [\cdot]_{\tau})
\]

### 4.2 Correctness of Translation

Given the translation \(\tau\), our goal is to state and prove the correctness of a translation of abstract tree triples \(\Omega \models \{p\} C\{q\}\) to concrete CAP triples \(\Omega' \models C_{\text{H}}\{p'\} \{C\}{q'}\) in the spirit of Pseudo-Theorem 1. To do this, we need to define a state translation between abstract tree states and concrete CAP states.

Recall from §3.3 that the universal quantification of context interface function \(I_{c}\) in Pseudo-Theorem 1 is incorrect, since interfaces captured by \(I_{c}\) are subject to change by the environment. We define the set of stable interface functions \(\text{SI}_{\tau}\) whereby an abstract address is associated with a single interface but with a set of interfaces. By associating a set of possible interfaces with an abstract address at any one time, we can account for the potential change of interfaces by the environment.

**Definition 4.4** Given the sets of inner and outer interfaces \(\text{IN}_{\tau}, \text{OUT}_{\tau}\) (def. 3.5), the set of stable inner-interface functions is \(\text{SI}_{\tau}^{\text{in}} : \mathcal{P}(\text{ADD} \rightarrow \mathcal{P}(\text{IN}_{\tau}))\) and the set of stable outer-interface functions is \(\text{SI}_{\tau}^{\text{out}} : \mathcal{P}(\text{ADD} \rightarrow \mathcal{P}(\text{OUT}_{\tau}))\). The set of stable interface functions is defined by \(\text{SI}_{\tau} \upmodels \text{SI}_{\tau}^{\text{in}} \times \text{SI}_{\tau}^{\text{out}}\). For \(SI \in \text{SI}_{\tau}\), \(SI^{\text{in}}\) and \(SI^{\text{out}}\) denote the first and second projections, respectively. The collapse of \(SI\) is \(SI \downmodels (SI^{\text{in}} \downarrow \times SI^{\text{out}} \downarrow)\) where \(SI^{\text{in}} \downarrow\) (and analogously \(SI^{\text{out}} \downarrow\)) is defined as:

\[
SI^{\text{in}} \downmodels \left\{ I \in \mathcal{T}_{\tau}^{\text{in}} \mid \text{dom}(I) = \text{dom}(SI^{\text{in}}) \land \forall x \in \text{dom}(I), I(x) \in SI^{\text{in}}(x) \right\}
\]

We can now give a state translation between abstract tree states and concrete CAP states, parametrised by the stable interface functions.

**Definition 4.5** The state translation: \(\llbracket \cdot \rrbracket_{\tau}^{(3)} : \mathcal{P}(\Sigma \times H_{\tau}) \rightarrow \text{SI}_{\tau} \rightarrow \mathcal{P}(\Sigma \times M_{\text{H}})\) is:

\[
\llbracket \langle p \rangle \rangle_{\tau}^{SI} \upmodels \left\{ (\sigma, h) \in p \land 3 I_{d}, m \in \bigcup_{I_{c} \in \text{SI}_{\tau}} \{ (h)_{\tau}^{I_{c}} \text{dom}(I_{d}^{\text{in}}) = h^{\text{in}} \land \text{dom}(I_{d}^{\text{out}}) = I_{d}^{\text{out}} \} \right\}
\]

167
where $i_{\text{in}}, i_{\text{out}}$ are given in Def. 2.4 and $I_1 \vert I_2 \triangleq (I_1^{\text{in}} \cup I_2^{\text{in}}, I_1^{\text{out}} \cup I_2^{\text{out}})$ with $\cup$ denoting the standard disjoint function union.

Note that when transforming a tree state, the variable store $\sigma$ remains unchanged while the tree heap $h$ is translated as $\langle h \rangle_{T}^{\text{H}d}$. $I_d$ captures the interfaces of abstract addresses within the control of the thread and thus are in the footprint of $h$. On the other hand, $I_e$ represents the interfaces of abstract addresses controlled by the environment. $I_d$ and $I_e$ are analogous to those of Pseudo-theorem 1, with the exception that they are drawn from the stable interface function $SI$.

We can now formalise the transformation of an abstract triple $\Omega \models T \{ p \} C \{ q \}$ to a concrete triple $\langle \langle \Omega \rangle \rangle_{T} \models \{ \{ p \} \} \{ \{ C \} \} \{ \{ q \} \}$, where $\langle \langle \cdot \rangle \rangle_{T}$ is defined in Def. 4.6.

**Definition 4.6** Given the library translation $\tau : T \to C_{\text{H}}$, for all procedure specification environments $\Omega \in \text{PEnv}_{T}$, abstract tree states $p, q \in \mathcal{V}(\Sigma \times H_{T})$ and tree programs $C \in \text{PROG}_{T}$, the translated triple $\tau : \Omega \models T \{ p \} C \{ q \}$ is:

$$\tau : \Omega \models T \{ p \} C \{ q \} \triangleq \forall SI \in SL_{T}. \{ \langle \langle \Omega \rangle \rangle_{T} \models \{ \{ p \}^{SI} \} \{ \{ C \} \} \{ \{ q \}^{SI} \} \}
$$

where $\langle \langle \Omega \rangle \rangle_{T} \models \{ f : \{ p \}^{SI} \models \{ q \}^{SI} | (f : p \models q) \in \Omega \land SI \in SL_{T} \}.$

We can now amend the statement of Pseudo-theorem 1 and formulate the theorem describing the correctness of the `deleteTree` command.

**Theorem 4.7** (deleteTree Correctness) The implementation of deleteTree command as given in Fig. 5 is correct if

$$\tau : \Omega \models T \{ \text{var}(n,n) \times \text{ATree}(n[t])(x) \} \text{ deleteTree } \{ \text{var}(n,n) \times \text{ATree}(\emptyset)(x) \}$$

**Proof.** The correctness proof of the above statement is given in the accompanying technical report [18].

**Definition 4.8** The library translation $\tau : T \to C_{\text{H}}$ is sound if, for all $\Omega \in \text{PEnv}_{T}$, $p, q \in \mathcal{V}(\Sigma \times H_{T})$ and $C \in \text{PROG}_{T}$:

$$\Omega \models T \{ p \} C \{ q \} \implies \tau : \Omega \models T \{ p \} C \{ q \}$$

**Theorem 4.9** (Sound translation) The library translation $\tau : T \to C_{\text{H}}$ is sound.

**Proof.** The proof is by induction over the structure of $C$ in $\Omega \models T \{ p \} C \{ q \}$ and is given in the technical report [18]. We show that translation $\tau$ has the following properties and appeal to these properties in the proof.

**Property 1 (Axiom Correctness)** For all $\Omega \in \text{PEnv}_{T}$, $p, q \in \mathcal{V}(\Sigma \times H_{T})$, $A \in \text{ATOM}_{T}$,

$$\Omega \models T \{ p \} A \{ q \} \implies \tau : \Omega \models T \{ p \} A \{ q \}$$

This ensures that the translation correctly implements abstract atomic commands, as demonstrated intuitively throughout the paper with the `deleteTree` command.

**Property 2 (Monotonicity of id Relation)** For all $h_1, h_2 \in H_{T}$ and $I \in \mathcal{I}_{T}$:

$$\{\langle h_1 \rangle \} \text{id} \{\langle h_2 \rangle \} \implies \{\langle h_1 \rangle^{I} \} \text{id} \{\langle h_2 \rangle^{I} \}$$

A lifting of this property to abstract tree states is used in proof of rule of consequence.
Property 3 (\(*\)-Preservation) For all \(h_1, h_2 \in H_T\) and \(I \in I_\tau\)
\[
\langle h_1 \bullet_T h_2 \rangle_\tau I \equiv \exists I_1, I_2. I_1 \cup I_2 = I \land \langle h_1 \rangle_\tau I_1 \bullet_C \langle h_2 \rangle_\tau I_2
\]

A lifting of this property to tree states is used in proof of disjoint concurrency rule.

Concluding Remarks. We have highlighted a gap in reasoning between abstract libraries and concrete implementations, due to the mismatch between the abstract connectivity of abstract data fragments and the concrete connectivity of their concrete representations. We have illustrated this gap using SSL reasoning applied to an abstract concurrent tree library \(T\) and CAP reasoning about a concrete implementation. This gap is closed by a refinement translation \(\tau: T \to C_H\), which depends crucially on an interface function \(I_\tau\) mapping abstract addresses to pointer interfaces. Our SSL reasoning and results generalise to arbitrary structured data libraries, as we report in [18]. Our work concentrates on the abstract connectivity associated with our SSL reasoning. However, the gap in reasoning occurs whenever there is a difference between abstract and concrete connectivity.

References

QRB, QFS, and the Probabilistic Powerdomain

Jean Goubault-Larrecq¹,²

LSV, ENS Cachan, CNRS, INRIA
ENS Cachan
61 avenue du président Wilson
94230 Cachan, France

Achim Jung¹,³

School of Computer Science
University of Birmingham
Birmingham, United Kingdom

Abstract

We show that the first author’s QRB-domains coincide with Li and Xu’s QFS-domains, and also with Lawson-compact quasi-continuous dcpos, with stably-compact locally limitary compact spaces, with sober QFS-spaces, and with sober QRB-spaces. The first three coincidences were discovered independently by Lawson and Xi. The equivalence with sober QFS-spaces is then applied to give a novel, direct proof that the probabilistic powerdomain of a QRB-domain is a QRB-domain. This improves upon a previous, similar result, which was limited to pointed, second-countable QRB-domains.

Keywords: QRB-spaces, QFS-spaces, QRB-domains, QFS-domains, stably compact spaces, probabilistic powerdomain

1 Introduction

An outstanding problem in denotational semantics is whether there is a full subcategory of continuous dcpos that is both Cartesian-closed and closed under the action of the probabilistic powerdomain monad \( V \) [17]. Indeed, there are very few categories of dcpos that are known to be closed under \( V \): the category of all dcpos, that of all continuous dcpos [15], and that of all Lawson compact continuous dcpos [17]. To that list, one must add the pointed, second-countable QRB-domains [10].

¹ We are grateful for the visiting professorship provided by ENS Cachan to the second author in March/April 2013, during which time the work reported here was begun.
² Email: goubault@lsv.ens-cachan.fr
³ Email: A.Jung@cs.bham.ac.uk
While QRB-domains are only quasi-continuous and not continuous domains, and do not form a Cartesian-closed category either, they have attracted considerable attention recently.

QRB-domains are defined by imitating RB-domains. Independently, Li and Xu used a similar process to define QFS-domains [23], imitating the construction of FS-domains [16]. Rather surprisingly, QRB and QFS-domains are the same thing (RB and FS-domains are not known to coincide), and are also exactly the Lawson-compact quasi-continuous domains. This was shown independently by the present authors and J. Lawson and X. Xi. We present our proof in Section 5 below; Lawson’s and Xi’s proof will appear as [21].

One of our characterizations of QRB is as so-called sober QFS-spaces, and this will turn out to be instrumental in proving that the category of all QRB-domains, and not just the second-countable ones, is closed under the action of the probabilistic powerdomain, as we shall see in Section 6. This improves upon [10], and relies on a rather different proof argument.

Outline. After some brief preliminaries (Section 2), we discuss the notion of functional approximation in Section 3. This is a central concept in domain theory, at the heart of RB-, FS-, QRB-, and QFS-domains. Another domain-theoretic leitmotiv is that one should always topologize (paraphrasing M. Stone), and we introduce QFS-spaces in Section 4 as the natural topological counterpart of QFS-domains. We give our proof that QRB-domains and QFS-domains are the same thing (and coincide with four other natural notions, including sober QFS-spaces) in Section 5. We apply this to the promised result that the probabilistic powerdomain of a QRB-domain is a QRB-domain in Section 6.

2 Preliminaries

We refer to the classic texts [6,1] for the required domain-theoretic background, and to [11] for topology.

We agree that a subset of a space is compact if and only if every open cover has a finite subcover, that is, we do not require separation. We take coherence to mean that the intersection of any two compact saturated subsets is compact. (A saturated subset is one that is equal to the intersection of its open neighborhoods.) A space is stably compact if it is sober, compact, locally compact and coherent. As is well-known, the patch topology of a stably compact space is compact Hausdorff, see [11, Section 9] or [6, Section VI-6] for more details.

Any sober space is well-filtered, meaning that if an open subset $U$ contains a filtered intersection $\bigcap_{i \in I} Q_i$ of compact saturated subsets, then $U$ contains $Q_i$ for some $i \in I$. In a well-filtered space, every such filtered intersection is compact.

Given a $T_0$ topological space $(X; \tau)$, we will make heavy use of its specialisation order defined as $x \leq y$ if $x \in \{y\}$. We write $\uparrow E$ for the upward closure (w.r.t. $\leq$) of a subset $E$. Subsets equal to their upward closure are exactly the saturated one. If $E$ is finite, then $\uparrow E$ is compact, and we call such sets the finitary compacts of $X$.

The set of compact saturated subsets of a topological space $(X; \tau)$ may be
equipped with an order by setting $A \leq B$ iff $A \supseteq B$, and we write $Q(X)$ for the resulting poset. It may also be equipped with the upper Vietoris topology, which has a base of opens of the form $\square U$, $U \in \tau$, where $\square U$ denotes the collection of compact saturated subsets contained in $U$. This yields the upper space $Q(V)(X)$ of $X$. Happily, the specialisation order of the upper space is precisely reverse inclusion. Analogously, we write $\text{Fin}(X)$ for the collection of finitary compacts of $X$, and topologize it with the subspace topology, yielding a space that we write $\text{Fin}(V)(X)$.

When $X$ is well-filtered (e.g., sober), $Q(X)$ is a dcpo and directed suprema are computed as intersections.

For a finite subset $E$ of a poset $(X, \leq)$ and $x \in X$, write $E \ll x$ iff every directed family $(x_i)_{i \in I}$ whose supremum $\sup_{i \in I} x_i$ is above $x$ in $X$ contains an element $x_i$ that is above some element $z$ of $E$. We also write $\uparrow E \ll x$ instead of $E \ll x$, stressing the fact that this is a property of the finitary compact $\uparrow E$, not just of the finite set $E$. The dcpo $X$ is a quasi-continuous domain (see [7] or [6, Definition III-3.2]) if and only if for every $x \in X$, the collection of all $\uparrow E \in \text{Fin}(X)$ that approximate $x$ ($\uparrow E \ll x$) is directed (w.r.t. $\supseteq$) and their least upper bound in $Q(X)$ is $\uparrow x$.

### 3 Functional approximation

We are concerned with spaces in which points are “systematically” approximated, by which we mean that we are given functions which produce approximants for each element. In domain theory, the idea goes back to Plotkin’s characterization of SFP-domains, [24], as those dcpos $X$ for which there is a chain of Scott-continuous functions $(\varphi_n)_{n \in \mathbb{N}}$ from $X$ to $X$, satisfying the following properties

(i) for each $n \in \mathbb{N}$, $\varphi_n \leq \text{id}_X$;

(ii) for each $n \in \mathbb{N}$, $\varphi_n \circ \varphi_n = \varphi_n$;

(iii) for each $n \in \mathbb{N}$, $\varphi_n$ has finite image;

(iv) $\text{id}_X = \bigvee_{n \in \mathbb{N}} \varphi_n$.

Plotkin also showed that the retracts of SFP-domains can be characterised similarly, by dropping the idempotency requirement (ii). If instead of a chain, only a directed family of such functions is present, then one obtains $RB$-domains. The concept was further generalized in the work of the second author, [16], where instead of requiring finite image, finite separation is stipulated: A function $\varphi : X \to X$ is finitely separated from $\text{id}_X$ if there exists a finite set $M \subseteq X$ such that

$$\forall x \in X. \exists m \in M. \varphi(x) \leq m \leq x.$$ 

An $FS$-domain, then, is a dcpo which contains a directed family $(\varphi_i)_{i \in I}$ of continuous functions finitely separated from identity such that $\text{id}_X = \bigvee_{i \in I} \varphi_i$.

In 2010 the first author, [9] realised that the concept of functional approximation could usefully be further generalized by allowing the approximating functions to produce compact neighborhoods rather than points, that is, the $\varphi_i$ now take values in $Q(V)(X)$ rather than $X$. With this generalization there are then two choices to be
Fig. 1. Two example spaces

made about their finiteness character:

**Choice 1** One can require the \( \varphi_i \) to have finite image in \( Q\mathcal{V}(X) \) or not make any such restriction.

**Choice 2** One can require the \( \varphi_i \) to only produce finitary compacts or allow general compact saturated sets.

Together this means that there are four variants that one might consider and it may come as a relief to the reader that they will in fact all turn out to lead to the same structures. Specifically, we will show that the most liberal notion, arbitrary image of general compact saturated sets, and the most restrictive one, finite image of finitary compacts, coincide. This will be true with and without assuming sobriety.

**Definition 3.1** A continuous function \( \varphi: X \to Q\mathcal{V}(X) \) is called a quasi-deflation if it is has finite image and for each \( x \in X, x \in \varphi(x) \in \text{Fin } X \). It is called quasi-finitely separated (or qfs for short) if there exists a finite set \( M \subseteq X \) such that for every \( x \in X \) there is \( m \in M \) such that \( x \in \uparrow m \subseteq \varphi(x) \). In this case, we say that \( \varphi \) is separated by \( M \), or that \( M \) is a separating set for \( \varphi \).

We shall agree to order continuous maps from \( X \) to \( Q\mathcal{V}(X) \) in the pointwise extension of \( \supseteq \). Accordingly, a family \( (\varphi_i)_{i \in I} \) of continuous functions from \( X \) to \( Q\mathcal{V}(X) \) is directed if and only if it is non-empty and for all \( i, j \in I \), there is a \( k \in I \) such that, for every \( x \in X \), \( \varphi_k(x) \subseteq \varphi_i(x), \varphi_j(x) \). We call it approximating if it is directed and furthermore, \( \uparrow x = \bigcap_{i \in I} \varphi_i(x) \) holds for all \( x \in X \).

We call a \( T_0 \) topological space \( (X; \tau) \) a QRB-space if there is an approximating family of quasi-deflations for it. It is called a QFS-space if there is an approximating family of quasi-finitely separated maps.

A QFS- (or QRB-) space \( (X; \tau; (\varphi_i)_{i \in I}) \) is called topological if for all \( U \in \tau \) and \( x \in U \) there is \( i \in I \) such that \( \varphi_i(x) \subseteq U \).

Clearly, every quasi-deflation \( \varphi \) is also qfs because we can take the finitely many minimal elements of the finitely many possible images of \( \varphi \) as the separating set. Therefore, every QRB-space is also QFS. To explain the last part of the definition, we give an example to show that not every QFS-space is topological:

**Example 3.2** Consider the poset \( P_1 \) in Figure 1 consisting of the natural numbers in their usual order plus an extra element \( a \) not related to any of the others. Equip this set with the Alexandroff topology (of all upper sets) and consider the map \( \varphi_m \)
which maps each \( n \in \mathbb{N} \) to \( \uparrow \min\{m, n\} \) and \( a \) to \( \uparrow m \cup \{a\} \). Clearly, each \( \varphi_m \) is a quasi-deflation. Furthermore, the family \((\varphi_m)_{m \in \mathbb{N}}\) is approximating and thus \( P_1 \) is a QRB-space. However, for no \( m \in \mathbb{N} \) do we have that \( \varphi_m(a) \subseteq \uparrow a = \{a\} \).

4 QFS-spaces

**Proposition 4.1** QFS-spaces are compact.

**Proof.** Let \( X \) be a QFS-space and \( \varphi \) be any qfs map on \( X \) with separating set \( M \). Then \( X = \uparrow M \) since every \( x \in X \) is above some \( m \in M \) by definition. Since \( M \) is finite, we have compactness. \( \square \)

For local compactness we start with a useful lemma:

**Lemma 4.2** Let \( \varphi \) be a qfs map on a topological space \( X \), separated by the finite set \( M \). Then for every \( x \in X \), \( x \) is in the interior of \( \uparrow (M \cap \varphi(x)) \).

**Proof.** Fix \( x \in X \) and let \( U = X \setminus \downarrow (M \setminus \varphi(x)) \). Because of the finiteness of \( M \), \( U \) is an open set and a neighborhood of \( \varphi(x) \). Let \( V = \varphi^{-1}(\uparrow U) \), which is an open set since \( \varphi \) is continuous. By construction, \( x \) is a member of \( V \) and, furthermore, we claim that \( V \subseteq \uparrow (M \cap \varphi(x)) \). Indeed, let \( y \in V \). Then \( \varphi(y) \subseteq U \) and hence the separating element \( m \in M \) with \( m \in \varphi(y) \) and \( m \leq y \) also belongs to \( U \). Hence \( m \in M \cap \varphi(x) \) and \( y \in \uparrow (M \cap \varphi(x)) \) follows. \( \square \)

A topological space \( X \) is **locally finitary compact** if every open neighborhood \( U \) of an arbitrary point \( x \) contains a locally finitary neighborhood \( \uparrow E \) of \( x \) that \( U \supseteq \uparrow E \supseteq \text{int}(\uparrow E) \ni x \). The notion originates with Isbell [14], and the \( T_0 \) such spaces are called qc-spaces in [21]. Every quasi-continuous domain is locally finitary compact, since in this case \( \text{int}(\uparrow E) = \{x \in X \mid E \ll x\} \) [6, III-3.6(ii)]. The following is immediate from the definitions and the preceding lemma:

**Lemma 4.3** Every topological QFS-space is locally finitary compact.

It would be nice if one could also show coherence for QFS-spaces but without further assumptions this is not possible, even for QRB-spaces:

**Example 4.4** Consider the poset \( P_2 \) in Figure 1 together with the Scott topology (note that the only non-trivial directed suprema are \( a = \bigvee_{n \in \mathbb{N}} a_n \) and \( b = \bigvee_{n \in \mathbb{N}} b_n \)). The QRB property is established by maps \( f_m, m \in \mathbb{N} \), which map

\[
\begin{align*}
a & \mapsto a_m \quad b \mapsto b_m \\
a_n & \mapsto a_{\min\{m, n\}} \quad b_n \mapsto b_{\min\{m, n\}} \quad c_n & \mapsto c_{\min\{m, n\}} \quad d_n \mapsto c_m \text{ for } n > m \\
da_n & \mapsto d_n \text{ for } n \leq m
\end{align*}
\]

and by setting \( \varphi_m(x) = \uparrow f_m(x) \). The resulting QRB-space is topological because every Scott neighborhood of \( a \) (resp. \( b \)) must contain some final segment of \( a_n \)'s (resp. \( b_n \)'s). It is not coherent, though, because \( \uparrow a \cap \uparrow b = \{d_n \mid n \in \mathbb{N}\} \) is not compact.
The situation is much nicer if we assume our spaces to be sober. First, since sobriety implies well-filteredness, we immediately have the following:

**Lemma 4.5** *Sober QFS-spaces are topologically.*

Combining the last two lemmas we get that sober QFS-spaces are locally finitary compact, and it is known from [3], or the equivalence between (6) and (11) in [22, Theorem 2], or [21, Corollary 3.6], or [11, Exercise 8.3.39], that the sober, locally finitary compact spaces are exactly the quasi-continuous dcpos in their Scott topology. Thus we have:

**Proposition 4.6** *Sober QFS-spaces are quasi-continuous domains, and their given topology coincides with the Scott topology derived from the specialisation order.*

Thus it is appropriate to call sober QFS-spaces, QFS-domains, and similarly for sober QRB-spaces. A little amount of work should convince the reader that these QFS-domains are exactly the same of those defined by Li and Xu [23].

How far are (topological) QFS-spaces from QFS-domains? As it turns out, not so far as we will now show that sobrification leads from one to the other.

The sobrification $\hat{X}$ of a topological space $(X; \tau)$ can be described in a number of ways; the most convenient for our purposes is to realise it concretely as the set of closed irreducible subsets of $X$, together with the topology $\hat{\tau}$ which consists of open sets $\hat{U} = \{ A \in \hat{X} \mid A \cap U \neq \emptyset \}$, where $U$ ranges over the open sets in $\tau$. Note that $X$ and $\hat{X}$ have isomorphic frames of opens.

Given a qfs map $\varphi : X \to \mathcal{Q}_V(X)$ we replace $\varphi(x)$ with its set of open neighborhoods, defined as $\{ U \in \tau \mid \varphi(x) \subseteq U \}$. This is always a Scott-open filter in the frame $\tau$, and the Hofmann-Mislove Theorem tells us that, conversely, every Scott-open filter $F$ of $\tau$ corresponds to a unique compact saturated set $Q_F$ of the sobrification $\hat{X}$ of $X$. Indeed, $F$ consists precisely of the opens $U$ such that $\hat{U}$ is a neighborhood of $Q_F$, that is, a closed irreducible set belongs to $Q_F$ if and only if it meets every member of $F$.

For the upper Vietoris topology on $\mathcal{Q}_V(X)$, the basic open set $\Box \hat{U}$ consists of those compacts $Q_F$ where $F$ ranges over the Scott-open filters which contain $U$. Using this setup, we define $\psi : \hat{X} \to \mathcal{Q}_V(X)$ by mapping $A \in \hat{X}$ to $Q_{\psi(A)}$ where $\psi(A) = \{ U \in \tau \mid \exists a \in A. \varphi(a) \subseteq U \} = \{ U \in \tau \mid A \cap \varphi^{-1}(\Box U) \neq \emptyset \}$.

**Lemma 4.7** *For $\varphi : X \to \mathcal{Q}_V(X)$ qfs, $\psi$ is a qfs map.*

**Proof.** The function $\hat{\varphi}$, equivalently $\psi$, is well-defined: if a directed union of opens belongs to $\psi(A)$ then it covers $\varphi(a)$ for some $a \in A$. Because $\varphi(a)$ is compact, one of them does so already. Filteredness follows from $\varphi^{-1}(\Box U \cap \Box V) = \varphi^{-1}(\Box(U \cap V)) = \varphi^{-1}(\Box U) \cap \varphi^{-1}(\Box V)$ and the assumption that $A$ is irreducible.

For continuity, observe that $\hat{\varphi}^{-1}(\Box \hat{U}) = \psi^{-1}(\{ F \mid U \in F \}) = \{ A \in \hat{X} \mid A \cap \varphi^{-1}(\Box U) \neq \emptyset \} = \varphi^{-1}(\Box U)$.

For finite separation, we assume that $M$ is a separating set for $\varphi$. We show that the set $\hat{M} = \{ \downarrow m \mid m \in M \}$ is separating for $\hat{\varphi}$. Let $A$ be a closed irreducible set.

---

4 A set is irreducible if it meets every member of a finite family of open sets precisely if it meets their intersection (from which it follows that irreducible sets are non-empty).
For every $U \in \psi(A)$ we have by definition that there is $a \in A$ such that $\varphi(a) \subseteq U$. It follows that $U \cap (M \cap A)$ is non-empty. Hence the family of these sets, indexed by $U \in \psi(A)$, is a proper filter on the finite set $M \cap A$ and so there is $m_A \in M$ belonging to all of them. We clearly have that $\downarrow m_A \subseteq A$ and because $m_A$ is in every $U \in \psi(A)$, $\downarrow m_A$ meets every element of $\psi(A)$, whence $\downarrow m_A \in Q_\psi(A) = \varphi(A)$. □

The above construction has been chosen for its brevity but we may point out that the underlying idea relies on a natural transformation $T$ (a “distributive law”) from $\lceil - \rceil \circ Q_V$ to $Q_V \circ \lceil - \rceil$. Our map $\varphi$ is the composition $\hat{X} \xrightarrow{\tilde{\varphi}} Q_\psi(\hat{X}) \xrightarrow{T} Q_V(\hat{X})$. An even more explicit construction is also possible, and it demonstrates nicely the usefulness of the “Topological Rudin Lemma” presented in [13]: We invite the reader to use the latter to show that $T(C) = \{ A \in \hat{X} \mid \forall Q \in C. Q \cap A \neq \emptyset \}$, and to also use it to reprove Lemma 4.7 with that definition.

Finally, we would like to show that the lifted family $(\varphi_i)_{i \in I}$ is approximating for $\hat{X}$. It is here where we need the condition that the original QFS space be topological, as without this condition this would not be the case. Consider again Example 3.2: The sobrification of the space $P_1$ consists of the sets $\downarrow x$, $x \in X$ plus one more, the chain $A = \mathbb{N}$. By definition, $A$ belongs to each $\varphi_m(\downarrow a)$: check that, for every $a' \leq a$, $A$ meets every open neighborhood $U$ of $\varphi_m(a')$. Hence $A$ is also in the intersection of all $\varphi_m(\downarrow a)$, but it does not belong to $\uparrow(\downarrow a) = \{ \{a\} \}$.

We come to the main result of this section:

**Theorem 4.8** The sobrification of a topological QFS space is a QFS domain.

**Proof.** All that remains is to show that the family $(\varphi_i)_{i \in I}$ is approximating for $\hat{X}$. Let $A \subseteq \hat{X}$ be a closed irreducible subset of $X$ and let $B$ be another such, not above $A$. This means that $B$ does not contain $A$ (as subsets of $X$), and so let $a \in A \setminus B$. By the definition of topological QFS spaces we obtain an index $i \in I$ such that $\varphi_i(a)$ is contained in the open set $X \setminus B$. Writing $\psi_i(A)$ for $\{ U \in \tau \mid \exists a \in A. \varphi_i(a) \subseteq U \}$, so that $\varphi_i(A) = Q_{\psi_i(A)}$, we obtain that $U \in \psi_i(A)$ for $U = X \setminus B$. Since $U$ does not meet $B$, $B$ is not in $\varphi_i(A)$. □

5 QFS-domains

We have already seen that the addition of sobriety to the conditions for a QFS-space results in much nicer structures. The best is still to come, however. We begin by giving a short argument to show that QFS-domains are coherent, a result which appears as Corollary 3.9 in [21]. First a lemma, also from [21]:

**Lemma 5.1** If $X$ is a QFS-domain then $Q_V(X)$ is an FS-domain.

**Proof.** If $\varphi$ is a qfs map on $X$ separated by $M$, then $\Phi: Q_V(X) \to Q_V(X)$, defined by $\Phi(K) = \uparrow \varphi[K]$, is finitely separated: For the separating set consider all sets $\uparrow E$, $E \subseteq M$. □

**Proposition 5.2** The topology of a QFS-domain is coherent.
Proof. Let \( K, L \) be compact saturated sets of \( X \). They are points in \( \mathcal{Q}_V(X) \) and generate principal upper, hence compact, sets \( \uparrow_{\mathcal{Q}_V(X)} K \) and \( \uparrow_{\mathcal{Q}_V(X)} L \). Since \( \mathcal{Q}_V(X) \) is an FS-domain, it is coherent, hence the set \( K = \uparrow_{\mathcal{Q}_V(X)} K \cap \uparrow_{\mathcal{Q}_V(X)} L \) is a compact saturated set. The claim follows from the observation that \( K \cap L = \bigcup K \) and the fact that \( \bigcup \), as the multiplication of the upper powerspace monad ([25, Chapter 7]), is a continuous map from \( \mathcal{Q}_V(\mathcal{Q}_V(X)) \) to \( \mathcal{Q}_V(X) \). \( \square \)

For quasi-continuous domains, compactness plus coherence is the same as compactness in the Lawson topology. This follows, for example, from the fact that the Lawson and patch topologies coincide on quasi-continuous dcpos [6, Lemma V-5.15], and that every patch-compact space is coherent and compact [11, Proposition 9.1.27], while conversely quasi-continuous domains are locally compact and sober [11, Exercise 8.2.15]. We thus have the following refinement of Proposition 4.6:

**Corollary 5.3** \( \mathcal{Q}FS \)-domains are Lawson-compact quasi-continuous domains equipped with their Scott topology.

We now work towards the converse of this:

**Proposition 5.4** Every compact, locally compact, coherent space \( X \) has an approximating family of maps \( \varphi_M : X \to \mathcal{Q}_V(X) \) with finite image. Precisely, \( M \) ranges over the finite \( \vee \)-semi-lattices \( M \) of compact saturated sets of \( X \), and \( \varphi_M \) maps each \( x \in X \) to the smallest element of \( M \) whose interior contains \( x \).

Note that \( \varphi_M \) takes values in \( \mathcal{Q}(X) \), not in \( \text{Fin}(X) \). Smallest is taken with respect to inclusion. A \( \vee \)-semi-lattice of compact saturated sets is a family of sets that is closed under finite intersections (in particular, contains \( X \)).

**Proof.** Define \( \varphi_M(x) \) as the intersection of all the elements \( Q \) of \( M \) that are neighborhoods of \( x \). Using the fact that \( M \) is finite, \( \varphi_M(x) \) is the smallest neighborhood of \( x \) in \( M \), so \( \varphi_M(x) \) is well defined, and in \( \mathcal{Q}(X) \) by coherence and compactness.

For continuity, let \( U \) be open and consider \( x \in \varphi_M^{-1}(\square U) \). Let \( Q = \varphi_M(x) \). For every \( y \in \text{int}(Q) \), \( \varphi_M(y) \subseteq Q \subseteq U \), so \( y \) is in \( \varphi_M^{-1}(\square U) \). Hence \( \text{int}(Q) \) is an open neighborhood of \( x \) included in \( \varphi_M^{-1}(\square U) \), so \( \varphi_M^{-1}(\square U) \) is open.

Clearly, if \( M \subseteq M' \), then \( \varphi_M(x) \supseteq \varphi_{M'}(x) \) for every \( x \in X \). The family of all \( \varphi_M \) is directed: given \( M \) and \( M' \), there is a smallest semi-lattice \( M \sqcup M' \) of compact saturated sets containing \( M \) and \( M' \), consisting of the intersections \( Q \cap Q' \) with \( Q \in M \) and \( Q' \in M' \); coherence implies that each such \( Q \cap Q' \) is compact saturated, and \( \varphi_{M \sqcup M'} \) is above both \( \varphi_M \) and \( \varphi_{M'} \) (w.r.t. \( \supseteq \)).

All that remains to show is that the maps \( \varphi_M \) form an approximating family. Given \( x \in X \), \( \uparrow x \subseteq \bigcap M \varphi_M(x) \) is by definition. For the reverse inclusion, we show that every open neighborhood \( U \) of \( x \) contains \( \bigcap M \varphi_M(x) \). By local compactness, \( U \) contains a compact saturated neighborhood \( Q \) of \( x \). \( M = \{Q, X\} \) qualifies as a semi-lattice of compact saturated sets, and we have \( \varphi_M(x) = Q \subseteq U \). \( \square \)

The following is standard:
Lemma 5.5 Let $X$ be a locally finitary compact space. For every compact saturated subset $Q$ of $X$, and every open neighborhood $U$ of $Q$, there is a further, finitary compact neighborhood $\uparrow E$ of $Q$ contained in $U$.

Proposition 5.6 Every compact, locally finitary compact, coherent space $X$ has an approximating family of quasi-deflations.

Proof. Applying Proposition 5.4, we obtain an approximating family of maps $\varphi_M$. We need to replace each compact saturated subset $Q \in \text{im } \varphi_M$ by a finitary compact.

Assume first that we are given an open neighborhood $U_Q$ around each of them. Lemma 5.5 allows us to find finitary compact neighborhoods $\uparrow E_Q$ between $Q$ and $U_Q$. We seek to find $\uparrow E_Q$ so that, additionally, $Q \subseteq Q'$ implies $\uparrow E_Q \subseteq \uparrow E_{Q'}$. To ensure this, we define $\uparrow E_Q$ step by step, always working on the largest $Q \in M$ that is still to be considered (so we start with $X$, the largest element of $M$). Given any $Q \in M$ such that $\uparrow E_{Q'}$ is already defined for every strictly larger $Q' \in M$, we apply Lemma 5.5 and define $\uparrow E_{Q'}$ as some finitary compact neighborhood of $Q$ contained in $\bigcup_{Q' \in M} \text{int}(\uparrow E_Q)$.

We now replace each $Q \in \text{im } \varphi_M$ by the so chosen $\uparrow E_Q$, resulting in a function $\psi_{M,\mathcal{E},\mathcal{U}}$, where $\mathcal{U}$ is the collection of open neighborhoods $U_Q$ we started with, and $\mathcal{E}$ is the collection of finitary compacts $\uparrow E_Q$. We need to check that $\psi_{M,\mathcal{E},\mathcal{U}}$ is continuous, and for that we check that $\psi_{M,\mathcal{E},\mathcal{U}}(\square U)$ is open for every open subset $U$ of $X$. Let $x$ be an element of $\psi_{M,\mathcal{E},\mathcal{U}}(\square U)$, and $Q = \varphi_M(x)$; in particular, $\uparrow E_Q \subseteq U$. As in the proof of Proposition 5.4, every element $y$ of $\text{int}(Q)$ is such that $\varphi_M(y) \subseteq Q$; for $Q' = \varphi_M(y)$, $Q' \subseteq Q$ implies $\uparrow E_{Q'} \subseteq \uparrow E_Q$, so $\psi_{M,\mathcal{E},\mathcal{U}}(y) \subseteq \uparrow E_Q \subseteq U$. Therefore $\text{int}(Q)$ is an open neighborhood of $x$ included in $\psi_{M,\mathcal{E},\mathcal{U}}(\square U)$.

The family of all maps $\psi_{M,\mathcal{E},\mathcal{U}}$ (namely, with $\mathcal{E} = (\uparrow E_Q)_{Q \in M}$ monotone, $\mathcal{U} = (U_Q)_{Q \in M}$, and $\mathcal{Q} \subseteq \text{int}(\uparrow E_Q) \subseteq \uparrow E_Q \subseteq U_Q$ for each $Q \in M$) is approximating, since we can choose the initial neighborhoods $U_Q$ as close to each $Q \in M$ as we like, and it remains to show that it is directed. It is non-empty: choose $M = \{X\}$ and $\mathcal{U} = M$, and define $\uparrow E_X$ as $X$ itself, which is finitary compact as a consequence of Lemma 5.5 with $Q = U = X$. We find an upper bound of $\psi_{M,\mathcal{E},\mathcal{U}}$ and $\psi_{M',\mathcal{E}',\mathcal{U}'}$ by defining $N = M \sqcup M'$, and for the open neighborhood system $\mathcal{V}$ we let $V_N = \bigcap \{\text{int}(\uparrow E_Q) \mid N \subseteq Q \in M\} \cap \bigcap \{\text{int}(\uparrow E_{Q'}) \mid N \subseteq Q' \in M'\}$ for each $N \in N$. (We write $\mathcal{E} = (\uparrow E_Q)_{Q \in M}$, $\mathcal{E}' = (\uparrow E_Q')_{Q' \in M'}$. It is clear that $\psi_{N,\mathcal{E},\mathcal{V}}$ is above $\psi_{M,\mathcal{E},\mathcal{U}}$ and $\psi_{M',\mathcal{E}',\mathcal{U}'}$.)

Theorem 5.7 Let $X$ be a topological space. The following are equivalent:

(i) $X$ is a stably compact, locally finitary compact space.
(ii) $X$ is a sober QRB-space.
(iii) $X$ is a sober QFS-space.
(iv) $X$ is a QRB-domain with its Scott topology.
(v) $X$ is a QFS-domain with its Scott topology.
(vi) $X$ is a Lawson-compact quasi-continuous dcpo in its Scott topology.
(vii) \( X \) is a compact, coherent, quasi-continuous dcpo in its Scott topology.

**Proof.** (i) \( \Rightarrow \) (ii): \( X \) is a QRB-space by Proposition 5.6, and sober since stably-compact. (ii) \( \Rightarrow \) (iii) and (iv) \( \Rightarrow \) (v) are obvious. (iii) \( \Rightarrow \) (v) is Proposition 4.6, which also implies (ii) \( \Rightarrow \) (iv) since QRB-spaces are instances of QFS-spaces.

(v) \( \Rightarrow \) (vi). Every QFS-domain is quasi-continuous [23, Proposition 3.8], and Lawson-compact [23, Theorem 4.9].

(vi) \( \Rightarrow \) (vii). For quasi-continuous domains, compactness plus coherence is the same as compactness in the Lawson topology.

(vii) \( \Rightarrow \) (i). Every quasi-continuous dcpo is sober [11, Exercise 8.2.15] and locally finitary compact [11, Exercise 5.2.31]. With compactness and coherence, this implies that \( X \) is stably-compact.

Lawson and Xi’s result mentioned in the introduction [21] is the equivalence (iv) \( \Leftrightarrow \) (v) \( \Leftrightarrow \) (vi) above. Items (i)–(iii) offer other, purely topological characterizations of QRB-domains.

Returning to the topological beginnings of our paper, we note the following:

**Theorem 5.8** Topological QFS-spaces are QRB.

**Proof.** Let \( (X; \tau; (\varphi_i)_{i \in I}) \) be a topological QFS-space. Then its sobrification \( \hat{X} \) is a QFS-domain and so by the preceding theorem, a QRB-domain. Looking at the proof of Proposition 5.6 we find that we constructed the finitary compacts by invoking Lemma 5.5, so we should have a closer look at that in the case that we are dealing with a locally finitary compact space that is a sobrification. In that case, every element \( e \) of \( E \) is a closed irreducible set \( A \) that meets the open set \( U \in \tau \). We may therefore replace \( e \) with the irreducible set \( \downarrow a \) where \( a \) is an arbitrarily chosen element of \( A \cap U \). In summary, then, we can make sure that the elements employed in the proof of Proposition 5.6 all stem from the image of the embedding \( x \mapsto \downarrow x \) of \( X \) into its sobrification. \( \square \)

6 The Probabilistic Powerdomain over QRB-Domains

Let us turn to the probabilistic powerdomain \( \mathcal{V}(X) \) over a space \( X \). This was introduced by Jones in her PhD thesis [15] to give semantics to higher-order programs with probabilistic choice. Jones proved that \( \mathcal{V}(X) \) was a continuous dcpo for every continuous dcpo \( X \), but also that \( \mathcal{V}(X) \) was not a continuous lattice, or even a bc-domain even for very simple continuous lattices or bc-domains \( X \). We still do not know whether \( \mathcal{V}(X) \) is an FS-domain, resp. an RB-domain whenever \( X \) is one, except in very specific cases [17]. However, the notion of functional approximation offered by QRB-domains is relaxed enough that the probabilistic powerdomain of a QRB-domain is again a QRB-domain. The first author proved this [10], for probability valuations (with total mass 1), and assuming second-countability.

Using Theorem 5.7, we shall see that the latter is an irrelevant assumption. We shall also prove it for spaces of continuous subprobability valuations, and of general,
unbounded, continuous valuations. The nature of the proof is very different from [10]: we build an approximating family of qfs maps on \( \mathcal{V}(X) \), directly.\(^5\)

The elements of \( \mathcal{V}(X) \) are a slight variation on the idea of a measure, and are called continuous valuations. A continuous valuation \( \nu \) on a space \( X \) is a Scott-continuous, strict, modular map from the complete lattice \( \mathcal{O}(X) \) of open subsets of \( X \) to \( \mathbb{R}^+ = \mathbb{R} \cup \{+\infty\} \). Strictness means that \( \nu(\emptyset) = 0 \), modularity states that \( \nu(U \cup V) + \nu(U \cap V) = \nu(U) + \nu(V) \) for all \( U, V \in \mathcal{O}(X) \).

Let \( \mathcal{V}(X) \), the probabilistic powerdomain over \( X \), denote the space of all continuous valuations on \( X \), with the weak topology. We also write \( \mathcal{V}^1(X) \) for the subspace of continuous probability valuations (\( \nu(X) = 1 \)) and \( \mathcal{V}^{\leq 1}(X) \) for the subspace of continuous subprobability valuations (\( \nu(X) \leq 1 \)). We shall write \( \mathcal{V}^\bullet(X) \) for \( \mathcal{V}(X) \), \( \mathcal{V}^1(X) \), or \( \mathcal{V}^{\leq 1}(X) \). The weak topology on \( \mathcal{V}^\bullet(X) \) has subbasic open sets of the form \([U > r], \) defined as \( \{ \nu \in \mathcal{V}^\bullet(X) \mid |\nu(U)| > r \} \) [19, Satz 8.5] (see also [12, Theorem 8.3]). Whatever \( \bullet \) is, \( \mathcal{V}^\bullet \) is a functor on the category of topological spaces, and its action \( \mathcal{V}f \) on continuous maps \( f: X \to Y \) is defined by \( \mathcal{V}f(\nu)(V) = \nu(f^{-1}(V)) \).

We again introduce a “distributivity law” \( \theta \), this time from \( \mathcal{V}^\bullet \mathcal{Q} \) to \( \mathcal{Q} \mathcal{V}^\bullet \). Given \( \mu \in \mathcal{V}^\bullet \mathcal{Q} \), one may define \( \theta(\mu) \) as the set of all \( \nu \in \mathcal{V}^\bullet(X) \) such that \( \nu(U) \geq \mu(\square U) \) for every open \( U \). It is not completely trivial that \( \theta(\mu) \) is non-empty and compact, or that \( \theta \) is continuous, but let us accept it for the moment. We may use \( \theta \) to produce maps \( \mathcal{V}^\bullet(X) \xrightarrow{\nu} \mathcal{V}^\bullet \mathcal{Q} \mathcal{V}^\bullet(X) \xrightarrow{\theta} \mathcal{Q} \mathcal{V}^\bullet(X) \) for an approximating family of quasi-deflations \( \varphi_i \) on \( X \). It will be fairly easy to see that the resulting maps are approximating, but they certainly do not have finite image, and even the image of a single \( \nu \in \mathcal{V}^\bullet(X) \) is in general not finitary. However, and up to a minor variation on the theme of \( \theta \) (\( \theta_f \), see below), we will manage to show that these maps are qfs. Hence \( \mathcal{V}^\bullet(X) \) will be a QFS space, and the equivalence between (iii) and (iv) of Theorem 5.7 will allow us to conclude.

**Lemma 6.1** Let \( (X; \tau) \) be a stably compact space. If \( \bullet \) is “1”, assume \( X \) pointed, too. For every Scott-continuous map \( f: \mathbb{R}^+ \to \mathbb{R}^+ \) such that \( f \leq \text{id}_{\mathbb{R}^+} \), the map \( \theta_f \) defined by \( \theta_f(\mu) = \{ \nu \in \mathcal{V}^\bullet(X) \mid \forall U \in \tau, \nu(U) \geq f(\mu(\square U)) \} \) is a continuous and Scott-continuous map from \( \mathcal{V}^\bullet(\mathcal{Q} \mathcal{V}(X)) \) to \( \mathcal{Q} \mathcal{V}(\mathcal{V}^\bullet(X)) \).

**Proof.** Let \( [X \to \mathbb{R}^+] \) denote the dcpo of Scott-continuous maps from \( X \) to \( \mathbb{R}^+ \), in the pointwise ordering. For any monotonic set function \( \xi \) on the open subsets of \( X \) with values in \( \mathbb{R}^+ \) and every \( h \in [X \to \mathbb{R}^+] \), one can define \( \int_{x \in X}^\infty h(x)d\xi \) by the Choquet formula \( \int_{t=0}^\infty \xi(h^{-1}(t, +\infty))dt \), where the latter is a Riemann integral. For \( \xi = \nu \in \mathcal{V}^\bullet(X) \), the functional \( h \mapsto \int_{x \in X} h(x)d\nu \) is Scott-continuous and linear [26, Section 4]. Scott-continuity follows from the fact that Riemann integration of antitone maps is itself Scott-continuous.

For \( \xi(U) = \mu(\square U) \), notice that \( h_s(Q) = \min_{x \in Q} h(x) \) defines a continuous function of \( Q \in \mathcal{Q}(X) \), since \( h_s^{-1}(t, +\infty] = \square h^{-1}(t, +\infty] \) (by compactness, the inf of \( h \) is attained on \( Q \)), so that \( \int_{x \in X} h(x)d\xi = \int_{t=0}^\infty \mu(\square h^{-1}(t, +\infty)]dt = \int_{t=0}^\infty \mu(h_s^{-1}(t, +\infty)]dt \).

\(^5\) This means proving the result using characterization 3 of QRB-domains given in Theorem 5.7. Characterization 1 may seem a better route, since \( \mathcal{V} \) is already known to preserve stable compactness: only local finitary compactness remains to be proved. However, that seems quite a formidable effort by itself.
and is therefore itself compact, since $\nu$ is superlinear, meaning that $p(h_1 + h_2) \geq p(h_1) + p(h_2)$ and $p(ah) = ap(h)$ for every $a \in \mathbb{R}^+$. Since $p(h) = \int_{x \in X} h(x) \, d\xi = \int_0^{+\infty} \xi(h^{-1}(t, +\infty)) \, dt$, $p$ is also Scott-continuous in $h$.

$\theta_f(\mu)$ is non-empty. Define $q : [X \rightarrow \mathbb{R}^+] \rightarrow \mathbb{R}^+$ by: $q(h) = \sup_{x \in X} h(x)$ if $\bullet$ is “1” or “≤ 1”, and $q(h) = +\infty \cdot \sup_{x \in X} h(x)$ otherwise, agreeing that $+\infty \cdot 0 = 0$. In each case, $q$ is sublinear ($q(h_1 + h_2) \leq q(h_1) + q(h_2)$), and $q(ah) = aq(h)$ for every $a \in \mathbb{R}^+$, and $p \leq q$. The space $[X \rightarrow \mathbb{R}^+]$ is a continuous dcpo, because $X$ is locally compact hence core-compact, and using Proposition 2 of [5], for example. Together with the obvious, pointwise addition and scalar multiplication by non-negative reals, $[X \rightarrow \mathbb{R}^+]$ is therefore a so-called continuous d-cone [27].

The Sandwich Theorem given there (Theorem 3.2) implies that there is a Scott-continuous linear map $\Lambda : X \rightarrow \mathbb{R}^+$ such that $p \leq \Lambda \leq q$. Defining $\nu(U)$ as $\Lambda(\chi_U)$, where $\chi_U$ is the characteristic map of $U$, yields a continuous valuation $\nu$ in $\mathcal{V}^\bullet(X)$ such that $\mu(\square U) = p(\chi_U) \leq \Lambda(\chi_U) = \nu(U)$ for every open subset $U$ of $X$.

Since $f \leq \id_{\mathbb{R}^+}$, in particular $\theta_f(\mu)$ is non-empty.

$\theta_f(\mu)$ is compact saturated. To show this, we use the following results. Define $\nu^\uparrow(Q)$, for $Q \in \mathcal{Q}(X)$, as $\inf_{Q \subseteq \mathcal{Q}} \nu(U)$, and $\langle Q \geq r \rangle$ as $\{ \nu \in \mathcal{V}^\bullet(X) | \nu^\uparrow(Q) \geq r \}$. The latter sets are compact saturated subsets of $\mathcal{V}^\bullet(X)$: this is a consequence of [8, Lemma 6.6] if $\bullet$ is “≤ 1” or “1”, and of [18, Theorem 6.5 (3)] otherwise.

We now notice that:

$$\theta_f(\mu) = \{ \nu \in \mathcal{V}^\bullet(X) | \forall Q \in \mathcal{Q}(X) \cdot \nu^\uparrow(Q) \geq a_Q^* \},$$

where $a_Q^* = \inf_{U \text{ open} \supseteq Q} f(\mu(\square U))$. Before we prove this, observe that $\theta_f(\mu)$ is therefore the intersection of the compact saturated subsets $\langle Q \geq a_Q^* \rangle$, $Q \in \mathcal{Q}(X)$, and is therefore itself compact, since $\mathcal{V}^\bullet(X)$ is stably compact. (The latter holds because $X$ is stably compact, see [17,2]. Technically, this is proved there for $\mathcal{V}^1(X)$ and $\mathcal{V}^{\leq 1}(X)$, but the proof is similar for $\mathcal{V}(X)$.)

To prove (1), let $a_U = f(\mu(\square U))$. Every $\nu \in \theta_f(\mu)$ trivially satisfies $\nu^\uparrow(Q) \geq a_Q^*$. Conversely, assume the latter holds for every $Q \in \mathcal{Q}(X)$. For every open subset $U$ of $X$, by local compactness $U$ is the directed union of all int$(Q)$, where $Q$ ranges over the compact saturated subsets of $U$. Since $\square$ commutes with directed unions, and $\mu$ and $f$ are Scott-continuous, $a_U = \sup_{Q \subseteq U} f(\mu(\square \text{int}(Q)))$. Since $\square \text{int}(Q) \subseteq U$ for every open neighborhood of $U$ of $Q$, this is less than or equal to $\sup_{Q \subseteq U} a_Q^*$, and the latter is less than or equal to $a_U$ because $a_Q^* \leq a_U$ whenever $Q \subseteq U$. Therefore $a_U = \sup_{Q \subseteq U} a_Q^*$. A similar argument shows that $\nu(\square U) = \sup_{Q \subseteq U} \nu^\uparrow(Q)$ (or see Tix [26, Satz 3.4 (1)]). It follows that $\nu(U) \geq a_U$. As $U$ is arbitrary, $\nu$ is in $\theta_f(\mu)$.

$\theta_f$ is Scott-continuous. Monotonicity is clear, while for a directed family $(\mu_i)_{i \in I}$ in $\mathcal{V}^\bullet(X)$, $\theta_f(\sup_{i \in I} \mu_i) = \{ \nu \in \mathcal{V}^\bullet(X) | \forall U \text{ open in } X \cdot \nu(U) \geq \sup_{i \in I} f(\mu_i(\square U)) \}$. (since $f$ is Scott-continuous) = $\bigcap_{i \in I} \theta_f(\mu_i)$.

$\theta_f$ is continuous. Since $X$ is $T_0$, well-filtered, and locally compact, $\mathcal{Q}(X)$ is a continuous dcpo, and the Scott and upper Vietoris topologies coincide [25, Section 7.3.4], i.e., $\mathcal{Q}(X) = \mathcal{Q}_U(X)$. The Kirch-Tix Theorem states that given a con-
continuous dcpo $Y$, the Scott and weak topologies coincide on $\mathcal{V}(Y)$ [26, Satz 4.10], and on $\mathcal{V}^\leq(Y)$ [19, Satz 8.6]: the same happens for $\mathcal{V}^1(Y)$ if additionally $Y$ is pointed, by a trick due to Edalat [4, Section 3]: $Y' = Y \setminus \{\perp\}$ is again a continuous dcpo, and $\mathcal{V}^1(Y)$ is isomorphic to $\mathcal{V}^\leq(Y')$. Taking $Y = \mathcal{Q}(X) = \mathcal{Q}_\nu(X)$ (and noticing that this is pointed, as $X$ is compact), we obtain that $\mathcal{V}^\bullet(\mathcal{Q}_\nu(X))$ has the Scott topology of the pointwise ordering. To show that $\theta_f$ is continuous, it is therefore enough to show that $\theta_f^{-1}(\square U)$ is open in the Scott topology for every open subset $U$ of $\mathcal{V}^\bullet(X)$. Since $\square U$ is itself Scott-open by well-filteredness, this amounts to the Scott-continuity of $\theta_f$. \hfill \Box

**Theorem 6.2** For every QRB-domain $X$, $\mathcal{V}(X)$, $\mathcal{V}^\leq(X)$, and also $\mathcal{V}^1(X)$ if $X$ is pointed, are QRB-domains.

**Proof.** Let $X$ be a QRB-domain, and $(\phi_i)_{i \in I}$ be an approximating family of quasi-deflations on $X$. By Theorem 5.7 (iii), we only need to show that $\mathcal{V}^\bullet(X)$ is a QFS-space. It is sober since stably compact, as we have noted earlier.

For $\epsilon \in (0, 1]$, and $t \in \mathbb{R}_+$, let $f_\epsilon(t) = \max(0, \min(t, 1/\epsilon) - \epsilon)$. This is a chain of Scott-continuous maps, as $\epsilon \geq \epsilon'$ implies $f_\epsilon \leq f_{\epsilon'}$. Also, $f_\epsilon \leq id_{\mathbb{R}_+}$. For short, write $\theta_\epsilon$ for the map $\theta_{f_\epsilon}$ given in Lemma 6.1, and define $\psi_\epsilon$ as $\theta_\epsilon \circ \varphi_i : \mathcal{V}^\bullet(X) \to \mathcal{Q}_\nu(\mathcal{V}^\bullet(X))$. The family $(\psi_\epsilon)_{i \in I, \epsilon \in (0, 1]}$ is directed, and for every $\nu \in \mathcal{V}^\bullet(X)$, we claim that $\bigcap_{i \in I, \epsilon \in (0, 1]} \psi_\epsilon(\nu) = \uparrow \nu$.

To this end, we notice that:

(a) $\bigcap_{\epsilon \in (0, 1]} \theta_\epsilon(\mu) = \theta_{id}(\mu)$. Indeed, $\bigcap_{\epsilon \in (0, 1]} \theta_\epsilon(\mu) = \{\nu \in \mathcal{V}^\bullet(X) \mid \forall U \text{ open in } X : \nu(U) \geq \sup_{\epsilon \in (0, 1]} f_\epsilon(\mu(\square U))\} = \{\nu' \in \mathcal{V}^\bullet(X) \mid \forall U \text{ open in } X : \nu'(U) \geq \mu(\square U)\} = \theta_{id}(\mu)$.

(b) $\bigcap_{\epsilon \in (0, 1]} \theta_{id}(\varphi_i(\nu)) = \uparrow \nu$. This is proved as follows. For every open subset $U$ of $X$, $\bigcup_{i \in I} \varphi_i^{-1}(\square U) = U$: the elements $x$ of $U$ are those such that $\uparrow x \in \square U$, and we obtained the desired equality by the defining property of quasi-deflations, plus well-filteredness. It follows that $\sup_{\epsilon \in (0, 1]} \nu(\varphi_i^{-1}(\square U)) = \nu(\bigcup_{i \in I} \varphi_i^{-1}(\square U)) = \nu(U)$. The elements $\nu'$ of $\bigcap_{\epsilon \in (0, 1]} \theta_{id}(\varphi_i(\nu))$ are those elements of $\mathcal{V}^\bullet(X)$ such that, for every $i \in I$, for every open subset $U$ of $X$, $\nu'(U) \geq \nu(\varphi_i(\nu))(\square U)$; equivalently, such that $\nu'(U) \geq \sup_{\epsilon \in (0, 1]} \nu(\varphi_i(\nu))(\square U) = \nu(U)$, and we conclude.

Using these, $\bigcap_{i \in I, \epsilon \in (0, 1]} \psi_\epsilon(\nu) = \bigcap_{i \in I} \bigcap_{\epsilon \in (0, 1]} \theta_\epsilon(\varphi_i(\nu)) = \bigcap_{i \in I} \theta_{id}(\varphi_i(\nu)) = \uparrow \nu$, as announced.

It remains to show that $\psi_\epsilon$ is qfs. Write $\delta_x$ for the Dirac mass at $x$, namely, the continuous valuation such that $\delta_x(U) = \chi_U(x)$ for every open $U$. Let $E$ be the finite set of all elements that are minimal in some finitary compact in the image of $\varphi_i$, $n$ be its cardinality, and let $M$ be the finite set of continuous valuations of the form $\sum_{x \in E} a_x \delta_x$, where each $a_x$ is an integer multiple of $\epsilon/n$ between 0 and $1/\epsilon$ (and with $\sum_{x \in E} a_x \leq 1$ if $\bullet$ is “$1$”, $\sum_{x \in E} a_x = 1$ if $\bullet$ is “$1$”). This will be our separating set.

Fix $\nu \in \mathcal{V}^\bullet(X)$. We first simplify the expression of $\psi_\epsilon(\nu)$. For $Q \in \mathcal{Q}(X)$, let Fig. 2: The function $f_\epsilon$.
\[ a_Q^* = \inf_{U \text{ open } \supseteq Q} f_\nu(V \varphi_i(\nu)(\square U)) = \inf_{U \text{ open } \supseteq Q} f_\nu(\nu(\varphi_i^{-1}(\square U))). \]

Let \( Q_1, \ldots, Q_m \) be the finitely many finitary compacts in the image of \( \varphi_i \). For \( J \subseteq \{1, \ldots, m\} \), write \( Q_J \) for \( \bigcup_{j \in J} Q_j \). We claim that \( \psi_\nu(\nu) = \bigcap_{J \subseteq \{1, \ldots, m\}} \langle Q_J \geq a_Q^* \rangle \). To this end, recall equality (1), which we have used in the course of proving Lemma 6.1: \( \theta(f(\mu)) = \{ \nu' \in V^*(X) \mid \forall Q \in Q(X) \cdot \nu'(Q) \geq \inf_{U \text{ open } \supseteq Q} f(\mu(\square U)) \} \). So \( \psi_\nu(\nu) = \{ \nu' \in V^*(X) \mid \forall Q \in Q(X) \cdot \nu'(Q) \geq a_Q^* \} = \bigcap_{Q \in Q(X)} \langle Q \geq a_Q^* \rangle \). Looking back at the definition of \( a_Q^* \), we see that, since \( \varphi_i \) has finite image, \( \varphi_i^{-1}(\square U) \) can only take finitely many values when \( U \) varies. The family of these values forms a (finite) filtered family of open sets, which therefore has a least element, which happens to be \( \varphi_i^{-1}(\square Q) \) (extending the \( \Box \) notation in the obvious way). Hence \( a_Q^* = f_\nu(\varphi_i^{-1}(\square Q)) \).

For every \( \nu' \in \bigcap_{J \subseteq \{1, \ldots, m\}} \langle Q_J \geq a_Q^* \rangle \), and every \( Q \in Q(X) \), let \( J \) be the set of indices \( j \in \{1, \ldots, m\} \) such that \( Q_j \subseteq Q \). Since \( \varphi_i \) takes its values among \( Q_1, \ldots, Q_m \), \( \varphi_i^{-1}(\square Q) = \varphi_i^{-1}(\square Q_j) \), so \( a_Q^* = a_Q^* \). It follows that \( \nu'(Q) \geq \nu'(Q_j) \geq a_Q^* \), and as \( Q \) is arbitrary, \( \nu' \in \bigcap_{Q \in Q(X)} \langle Q \geq a_Q^* \rangle = \psi_\nu(\nu) \). The converse inclusion \( \psi_\nu(\nu) \subseteq \bigcap_{J \subseteq \{1, \ldots, m\}} \langle Q_J \geq a_Q^* \rangle \) is obvious.

To show that \( \psi_\nu \) is qfs, it will therefore be enough to find an element \( \sum_{x \in E} a_x \delta_x \) of \( M \) below \( \nu \) and in \( \psi_\nu(\nu) = \bigcap_{J \subseteq \{1, \ldots, m\}} \langle Q_J \geq a_Q^* \rangle \).

Let \( L \) be the finite lattice of all intersections of sets of the form \( \uparrow_A, A \subseteq E \). Tix observed that \( \nu(\cdot) \) defined a valuation on the compact saturated subsets of \( X \) [26, Satz 3.4 (2–4)]. In particular \( \nu(\cdot) \) restricts to a valuation on \( L \). Using the Smiley-Horn-Tarski Theorem (see, e.g., [20, Theorem 3.4]), \( \nu(\cdot) \) extends to an additive measure on the algebra \( \rho L \) of subsets generated by \( L \). The algebra \( \rho L \) is the smallest collection of subsets containing \( L \) and closed under unions, intersections, and complements. Its elements are the finite disjoint unions of sets of the form \( \bigcap_{x \in A} \uparrow x \cap \bigcup_{x \in E \setminus A} \uparrow x, A \subseteq E \).

For each \( x \in E \), let \( b_x = \nu(\uparrow_A) \) where \( A_x \) is the unique subset of \( E \) such that \( A_x \) contains \( x \). This definition ensures that \( \nu(\cdot)(B) = \sum_{x \in E \cap \uparrow_A} b_x = \sum_{x \in E} b_x \delta_x(B) \) for every \( B \in L \). For every open subset \( U \) of \( X \), and every \( x \in E \cap \uparrow_A \), \( C_{A_x} \subseteq \uparrow x \subseteq U \), so \( (\sum_{x \in E} b_x \delta_x(U) = \sum_{x \in E \cap \uparrow_A} \nu(\uparrow_A) = \nu(\bigcup_{x \in E \cap \uparrow_A} \uparrow_A) \) (since the sum is disjoint) \( \leq \nu(\uparrow_A \cap U) \leq \nu(U) \).

When \( \bullet \) is “\( \leq 1 \)”, we define the desired element \( \sum_{x \in E} a_x \delta_x \) of \( M \) by letting \( a_x \) be the nearest multiple of \( \epsilon \) below \( b_x \), namely \( \frac{\epsilon}{\delta} \cdot \lfloor \frac{\epsilon}{\delta} b_x \rfloor \). Clearly, \( \sum_{x \in E} a_x \delta_x \leq \sum_{x \in E} b_x \delta_x \leq \nu. \) Moreover, for every \( J \subseteq \{1, \ldots, m\} \), \( \sum_{x \in E} a_x \delta_x, \nu, (\sum_{x \in E} b_x \delta_x)(Q_j) = \sum_{x \in E \cap Q_j} a_x + \epsilon \geq \sum_{x \in E \cap Q_j} \epsilon x + \frac{\epsilon}{\delta} b_x \) since \( Q_j \) belongs to \( \rho L \). Since \( \varphi_i(x) \geq \uparrow x \) for every \( x \in X \), \( \varphi_i^{-1}(\square Q_j) \subseteq Q_j \). For every \( Q \in Q(X) \), recall that \( \varphi_i^{-1}(\square Q) \) is the least element of some finite filtered family of open sets, hence is open. It follows that the notation \( \nu(\varphi_i^{-1}(\square Q_j)) \) makes sense. From \( \varphi_i^{-1}(\square Q_j) \subseteq Q_j \), we obtain \( \nu(\varphi_i^{-1}(\square Q_j)) \geq \nu(\varphi_i^{-1}(\square Q_j)) \). We have just shown that \( \sum_{x \in E} a_x \delta_x(\varphi_i^{-1}(\square Q_j)) = \sum_{x \in E} b_x \delta_x(\varphi_i^{-1}(\square Q_j)) = \sum_{x \in E \cap \uparrow_A} \nu(\uparrow_A) - \epsilon = \nu(\varphi_i^{-1}(\square Q_j)) \geq 0 \). (We are silently using the fact that \( \nu(\uparrow_A) = \max(0, \nu(\varphi_i^{-1}(\square Q_j))) \) is \( \nu(\varphi_i^{-1}(\square Q_j)) \). Therefore \( \sum_{x \in E} a_x \delta_x \) is in \( \langle Q_J \geq a_Q^* \rangle \), and as \( J \) is arbitrary, it is in \( \psi_\nu(\nu) \).

When \( \bullet \) is “\( 1 \)” instead, we use the standard trick of putting all the missing mass on the bottom element \( \perp \). In other words, we define \( a_x \) as above for \( x \neq \perp \), and
as $1 - \sum_{x \in E, x \neq \bot} a_x$ otherwise. (Note that $E$ contains $\bot$. Indeed, it appears as the minimal element of $\varphi_i(\bot) = X$.) We check that $(\sum_{x \in E} a_x \delta_x)(U) \leq \nu(U)$ as above when $U$ does not contain $\bot$, while the same inequality reduces to the trivial $1 \leq 1$ when $U$ contains $\bot$, namely when $U = X$. Since we are using larger coefficients $a_x$ than in the “$\leq 1$” case, the fact that $(\sum_{x \in E} a_x \delta_x)\dagger(J) \geq a^*_Q J$, follows by the same arguments. It follows, again, that $\sum_{x \in E} a_x \delta_x$ is in $\bigcap_{j \in \{1,dots,m\}} (Q_J \geq a^*_Q J) = \psi_k(\nu)$.

Finally, when $\bullet$ is neither “$\leq 1$” not “$1$”, we argue as in the “$\leq 1$” case, except we now define $a_x$ as $\frac{1}{n} \frac{\epsilon}{x} \min(\frac{1}{\epsilon}, b_x)$. To check that $(\sum_{x \in E} a_x \delta_x)\dagger(J) \geq a^*_Q J$, we compute $(\sum_{x \in E} a_x \delta_x)\dagger(J) + \epsilon = \sum_{x \in E \cap Q_J} a_x + \epsilon \geq \sum_{x \in E \cap Q_J}(a_x + \frac{\epsilon}{n}) \geq \sum_{x \in E \cap Q_J, \min(\frac{1}{\epsilon}, b_x)}. If every $b_x$ is less than or equal to $\frac{1}{\epsilon}$, the latter is equal to $\nu(\hat{Q}_J)$, hence greater than or equal to $\min(\frac{1}{\epsilon}, \nu(\hat{Q}_J))$. If $b_x > \frac{1}{\epsilon}$ for some $x \in E \cap Q_J$, then $\nu(\hat{Q}_J) \geq \nu(\hat{C}_A) = b_x > \frac{1}{\epsilon}$, so $\sum_{x \in E \cap Q_J} \min(\frac{1}{\epsilon}, b_x) \geq \frac{1}{\epsilon} = \min(\frac{1}{\epsilon}, \nu(\hat{Q}_J)). In any case, $(\sum_{x \in E} a_x \delta_x)\dagger(J) + \epsilon \geq \min(\frac{1}{\epsilon}, \nu(\hat{Q}_J))$. As in the “$\leq 1$” case, this implies $(\sum_{x \in E} a_x \delta_x)\dagger(J) + \epsilon \geq \min(\frac{1}{\epsilon}, \nu(\hat{Q}_J))$, so $(\sum_{x \in E} a_x \delta_x)\dagger(J) \geq f_\epsilon(\nu(\varphi^{-1}_i(\hat{Q}_J))) = a^*_Q J$.

References


Adequacy and Complete Axiomatization for Timed Modal Logic

Samy Jaziri\textsuperscript{1}  Kim G. Larsen\textsuperscript{2}  Radu Mardare\textsuperscript{2}  Bingtian Xue\textsuperscript{2}

\textsuperscript{1}Lab. Spé cification & Vérification, ENS Cachan, France \quad \textsuperscript{2}Aalborg University, Denmark

Abstract

In this paper we develop the metatheory for Timed Modal Logic (TML), which is the modal logic used for the analysis of timed transition systems (TTSs). We solve a series of long-standing open problems related to TML. Firstly, we prove that TML enjoys the Hennessy-Milner property and solve one of the open questions in the field. Secondly, we prove that the set of validities are not recursively enumerable. Nevertheless, we develop a strongly-complete proof system for TML. Since the logic is not compact, the proof system contains infinitary rules, but only with countable sets of instances. Thus, we can involve topological results regarding Stone spaces, such as the Rasiowa-Sikorski lemma, to complete the proofs.

Keywords: Non-compact modal logics, complete axiomatization, timed modal logic.

1 Introduction

In the areas of embedded and cyber-physical systems, more than two decades of research have been dedicated to developing quantitative modeling and specification formalisms that allow for the construction of systems with guaranteed functional and non-functional properties. In particular, many embedded systems are highly safety critical, with hard constraints on real-time behaviors being essential. Here, the notion of timed transition system and timed automata [4] have proven extremely convenient for modeling purposes and are now routinely used for the analysis of real-time communication protocols and control programs.

Accompanying the timed transition systems, a variety of timed temporal logics have been introduced as convenient ways of capturing requirements to real-time systems. These logics include MTL [13], MITL [5], TPTL [7, 8] and ECL [21] all providing timed extensions of LTL. Similarly TCTL [2], T\textsubscript{\(\mu\)} [12] and L\textsubscript{\(\nu\)} [16] provide timed extensions of CTL and the modal \(\mu\)-calculus. Emphasis has been on detailed investigation of decidability and complexity of model checking and satisfiability checking,
identifying the importance of (absence of) punctual constraints in the logic [5, 15] as well as of the choice of semantics of timed automata (point-wise or continuous) [17–19]. Also, with the purpose of gaining decidability, satisfiability checking given constraints on number of clocks and size of constants in a satisfying timed automata has been considered [16]. In all of the aforementioned logics, quantitative requirements are either obtained by decorating modalities with timing constraints, or by using clocks in formulas.

The presence of time in all these settings makes it difficult to address fundamental meta-theoretical questions regarding the timed logics, such as adequacy\(^1\) or the development of complete proof systems. Regarding the axiomatization, there exists a series of results proved in very restricted settings, such as for TPTL [11] that only looks to models with discrete time, or for ECTL [22] under the restriction of models with finite variability – only finitely many state changes can occur in a finite amount of time. Other attempts, such as [9], failed already in achieving soundness results, as argued in [22].

In this paper we take the challenge of developing a strongly-complete proof system for the most basic Timed Modal Logic (TML) defined for the most general model of real-time systems, without any restriction on the nature of time. Our models are timed labelled transition systems (TTSs), which generalize timed automata [1]: their transitions are labeled with actions or time delays (real numbers). Our logic is the non-recursive fragment of \(L_\nu\) [16], which generalizes the logic of [1]. In these settings we solve a series of open problems.

Firstly, we prove that TML is adequate not only for timed automata, but in general, for entire class of TTSs. This settles an open problem and disproves a belief often found in the literature, e.g., [1] – that such a logic is not sufficiently expressive to characterize timed-bisimulation. In proving this result, we use a novel exploitation of formulas with free clock variables.

Secondly, we prove that the satisfiability problem for TML is undecidable despite its restrictive expressive power, thus generalizing the known undecidability result of satisfiability for TCTL [2]. Moreover, we show that the set of TML-validities is not recursively-enumerable. This implicitly means that any complete proof system will generate a non-recursively enumerable set of provable formulas, and makes one wonder whether this logic can, in fact, be axiomatized at all - see e.g., the discussion in [6].

We prove that TML can be axiomatized and we develop a proof system for it that is strongly-complete\(^2\) for the TTS-semantics. However, TML is not compact: there exists an infinite set of formulas that admits no model while all its finite subsets have models. For this reason our axiomatization must contains infinitary proof rules; they reflect Archimedean properties of the rational numbers used to interpret

---

\(^1\) A logic is adequate when its semantical equivalence over the class of models coincides with bisimilarity.

\(^2\) Strong completeness means that \(\Phi \vdash \phi \iff \Phi \models \phi\), where \(\Phi \models \phi\) denotes that all the models of the set \(\Phi\) of formulas are also models of the formula \(\phi\), and \(\Phi \vdash \phi\) denotes that \(\phi\) is provable from \(\Phi\).
clocks. Due to these infinitary rules, the proofs of our system cannot be enumerated. Nevertheless, these rules have countable sets of instances.

Our axioms provide a set of sufficient conditions that characterizes the concept of time in models like TTSs and timed automata. They reflect properties such as persistence, continuity, linearity, density of time, synchronization, or the fact that all clocks measure the same time flow. Most of these axioms are similar to axioms seen in other modal logics and these relations open interesting further questions.

An other important contribution of the paper is the construction of the canonical model, which generalizes for the case of a (non-compact) higher-order modal logic the classic filtration construction usually used for propositional modal logics. This follows the line opened by the second and the third author in collaboration with Panangaden and Kozen in [14]; it involves complex topological and model-theoretical arguments, such as the Rasiowa-Sikorski lemma, which are essential in achieving the main results and are pointing to a general methodology for constructing canonical models for non-compact modal logics.

This paper does not aim at addressing problems related to timed automata, nor to model verification, model construction or model checking. Our purpose is to clarify the open problems of model theory for TML. The interesting questions regarding timed automata-semantics will be addressed in a future work.

2 Preliminaries

Hereafter we fix the notation used in the paper.

Kleene equality. Given a partial function \( f : X \rightarrow Y \), we write \( f(x) \approx f(x') \) for \( x, x' \in X \) to denote the fact that \( f(x) \) and \( f(x') \) are simultaneously well-defined and whenever they are well-defined, they are equal.

Orders on reals. We use \( \preceq \) and \( \succeq \) to range over the set \( \{\leq, \geq\} \) such that \( \{\preceq, \succeq\} = \{\leq, \geq\} \); this means that \( \preceq \) can either be interpreted as \( \leq \) or as \( \geq \); if \( \preceq \) represents one of the two, then \( \succeq \) denotes the other. Similarly, we use \( < \) and \( > \) to range over the set \( \{<, >\} \) such that \( \{<, >\} = \{<, >\} \). Moreover, \( x < y \) means \( x \leq y \) and \( x \neq y \), and similarly for \( > \) and \( \geq \). We use \( \bowtie \) to denote to an arbitrary element of the set \( \{\leq, \geq, <, >\} \).

Interpretations on reals. Given a set \( \mathcal{K} \), an interpretation of \( \mathcal{K} \) (on non-negative reals) is a function \( i : \mathcal{K} \rightarrow \mathbb{R}_{\geq 0} \); if \( x \in \mathcal{K} \) and \( r \in \mathbb{R}_{\geq 0} \), we denote by \( i[x \mapsto r] \) the interpretation \( j \) of \( \mathcal{K} \) such that \( j(x) = r \) and \( j(y) = i(y) \) for \( y \neq x \). The arithmetic operations on interpretations are defined pointwise; 0 is used for the constant null interpretation and \( [x \mapsto r] \) denotes \( 0[x \mapsto r] \).

The Rasiowa-Sikorski Lemma [10, 20] is a result with important applications in logic.
Definition 2.1 Let $B$ be a Boolean algebra and let $T \subseteq B$ such that $T$ has a greatest lower bound $\bigwedge T$ in $B$. An ultrafilter (maximal filter) $U$ is said to respect $T$ if $T \subseteq U$ implies that $\bigwedge T \in U$.

If $\mathcal{T}$ is a family of subsets of $B$, we say that an ultrafilter $U$ respects $\mathcal{T}$ if it respects every member of $\mathcal{T}$.

Lemma 2.2 (Rasiowa–Sikorski [20]) For any Boolean algebra $B$ and any countable family $\mathcal{T}$ of subsets of $B$, each member of which has a meet in $B$, and for any nonzero $x \in B$, there exists an ultrafilter in $B$ that contains $x$ and respects $\mathcal{T}$.

3 Timed Transition Systems

A timed transition system (TTS) [1] is a labeled transition system that uses both actions and time delays as transition labels. The delay transitions describe the time flow and consequently are continuous and deterministic. Here we propose an equivalent definition for TTS that encodes the time in an algebraic format and simplifies our future developments.

Definition 3.1 [Timed Transition System] A timed labeled transition system (TTS) is a tuple $W = (M, \Sigma, \theta, \oplus)$ where $M$ is a non-empty set of states, $\Sigma$ a non-empty set of actions, $\theta : M \times \Sigma \rightarrow 2^M$ is the action-labelling function and $\oplus : \mathbb{R}_{\geq 0} \times M \rightarrow M$ is a partial function that encodes the delay transitions; for arbitrary $m \in M$ and $d, d' \in \mathbb{R}_{\geq 0}$,

1. $0 \oplus m = m$;
2. $d \oplus (d' \oplus m) \simeq (d + d') \oplus m$.

Whenever it is defined, $d \oplus m$ denotes a time delay $d$ applied to the state $m$. Condition 1 guarantees that a zero-delay is always well-defined and it does not change the state of the system; and condition 2 expresses that time is both additive and deterministic. As usual, instead of $m' \in \theta(m, a)$ we will write $m \xrightarrow{a} m'$.

In the rest of this paper we fix the set $\Sigma$ and omit it in the description of TTSs.

A timed bisimulation is a relation that equates states of a TTS with identical behaviours.

Definition 3.2 [Timed Bisimulation] Given a TTS $W = (M, \oplus)$, a timed bisimulation is an equivalence relation $R \subseteq M \times M$ such that whenever $(m, n) \in R$, for all $a \in \Sigma$ and $d \in \mathbb{R}_{\geq 0}$,

- if $m \xrightarrow{a} m'$, then there exists $n' \in M$ s.t. $n \xrightarrow{a} n'$ and $(m', n') \in R$;
- if $d \oplus m$ is well-defined, then $d \oplus n$ is well-defined and $(d \oplus m, d \oplus n) \in R$.

As for the other types of bisimulation, the previous definition can be extended to define the time bisimulation between distinct TTSs by considering bisimulation relations on their disjoint union. Time bisimilarity is the largest time-bisimulation
relation; if \( W_i = (M_i, \theta_i, \oplus_i) \), \( m_i \in M_i \) for \( i = 1, 2 \) and \( m_1 \) and \( m_2 \) are bisimilar, we write \((m_1, W_1) \sim (m_2, W_2)\).

### 4 Timed Modal Logic

In this section we introduce the Timed Modal Logic (TML) that encodes properties of TTSs. It is defined for a countable set \( K \) of clocks that we consider fixed in what follows. It contains Hennessy-Milner operators \([a] \phi\) for the actions \( a \in \Sigma\), where \( \Sigma \) is the fixed set of actions for which we have defined TTSs in the previous section. In addition, it is endowed with time inequalities of type \( x \sqsubseteq r \) for rational values \( r \) that evaluate the clock \( x \in K \) at the current state; with delay quantifiers \( \forall \forall \phi \) that predicate properties for any time-delay of the current state; and clock quantifiers \( \forall x.\phi \) that predicate properties for any interpretation of the clock \( x \) at the current state.

**Definition 4.1** [Syntax] For arbitrary \( r \in \mathbb{Q}_{\geq 0} \) and \( a \in \Sigma\),

\[
\mathcal{L} : \quad \phi := \bot | x \sqsubseteq r | \phi \rightarrow \psi | [a] \phi | \forall \forall \phi | \forall x.\phi.
\]

Let \( \Im(K) \) be the set of interpretations of \( K \). The semantics of TML is defined for an arbitrary TTS \( M = (M, \theta, \oplus) \), \( m \in M \) and \( i \in \Im(K) \) as follows.

- \( M, m, i \models \bot \) – never;
- \( M, m, i \models x \sqsubseteq r \) if \( i(x) \leq r \);
- \( M, m, i \models \phi \rightarrow \psi \) if \( M, m, i \models \psi \) whenever \( M, m, i \models \phi \);
- \( M, m, i \models [a] \phi \) if [for any \( m' \in M \) s.t. \( m \xrightarrow{a} m' \), \( M, m', i \models \phi \)];
- \( M, m, i \models \forall \forall \phi \) if [for any \( d \in \mathbb{R}_{\geq 0} \) s.t. \( d \oplus m \) is well-defined, \( M, d \oplus m, i + d \models \phi \)];
- \( M, m, i \models \forall x.\phi \) if [\( \forall t \in \mathbb{R}_{\geq 0}, M, m, i[x \mapsto t] \models \phi \)]

We also use, in addition, all the boolean operators defined as usual and the De Morgan duals of the modal operators: \( \langle a \rangle \phi \overset{df}{=} \neg[a] \neg \phi, x \triangleright r \overset{df}{=} \neg (x \leq r), \exists \phi \overset{df}{=} \neg (\forall \neg \phi), \) and \( \exists x.\phi \overset{df}{=} \neg (\forall x.\neg \phi) \). Other derived operators used in what follows are:

\( (x = r) \overset{df}{=} (x \leq r) \land (x \geq r) \) and \( \forall (x \triangleright r).\phi \overset{df}{=} \forall x.(x \triangleright r \rightarrow \phi) \), \( \triangleright \in \{\leq, \geq, <, >\} \).

In TML we can express the reset operator used in \([1, 12, 16]\) by

\[
x \text{ in } \phi \overset{df}{=} \forall x.(x = 0 \rightarrow \phi).
\]

Whenever it is not the case that \( M, m, i \models \phi \), we write \( M, m, i \not\models \phi \). We say that a formula \( \phi \) is satisfiable if there exists at least one TTS that satisfies it in one of its states under at least one interpretation; \( \phi \) is a validity if it is satisfied in all states of any TTS under any interpretation - in this case we write \( \models \phi \). For an arbitrary set \( \Phi \subseteq \mathcal{L} \), we write \( \Phi \models \phi \) if all the models of all the formulas in \( \Phi \) are also models of \( \phi \).
4.1 Undecidability of TML.

Individual formulae of TML express properties which only depend on the behaviour of a TTS up to a finite action-depth, thus making TML significantly less expressive than TCTL [3]. In a number of papers it has been shown how recursive extensions of TML – e.g. extensions with the ability to define logical properties recursively – enable the encoding of TCTL, while maintaining decidability of model-checking.

As stated in the theorem below, despite its limited expressive power the question of satisfiability for TML is (highly) undecidable, as is the case of TCTL. Formally, given an arbitrary TML formula $\phi \in L$, it is undecidable whether there exists a TTS $(M, \theta, \oplus)$ with $m \in M$ and interpretation $i \in \mathcal{I}(K)$ such that $M, m, i \models \phi$.

**Theorem 4.2 (Undecidability of TML)** The satisfiability question for TML is $\Sigma_1^1$-hard, hence undecidable.

**Proof.** We show that we can reduce the TML satisfiability question into the question as to whether a non-deterministic 2-counter machines has a computation with the initial location being visited infinitely often. This last question is known to be $\Sigma_1^1$-hard. Our proof is similar to the one in [3] and for this reason it is presented in the Appendix.

The undecidability of satisfiability for TML implies, as usual, the undecidability of validity for TML. In fact, Theorem 4.2 proves that the set of validities is not recursively enumerable.

5 Adequacy of TML

The Hennessy-Milner property (H-Mp), which states for a logic that bisimilarity of the models coincides with the semantic equivalence induced by the logic, is currently an open problem for timed logic. In [1] it was proven that the closed formulas (without free clock variables) cannot characterize bisimilarity. In this section we prove the H-Mp for TML, therefore we solve the adequacy problem.

Before proceeding with the proof, observe the essential role of interpretations in the semantics of TML. Consider the two TTSs depicted in Figure 1, where the horizontal lines represent the time flow from the initial states $m$ and $m'$ respectively. The two systems in the initial states can delay forever and they can both take an $a$-transition to states that satisfy $END$ after each delay $2 - \frac{2}{2^n}$ and $2 + \frac{2}{2^n}$ for each integer $n > 0$. However, the two systems differ: $m$ can take an $a$-transition after the delay 2, while $m'$ cannot.

If we consider an interpretation $i \in \mathcal{I}(K)$ s.t. $i(x) \notin Q_{\geq 0}$ for any $x \in K$, one can notice that $m$ and $m'$ satisfy exactly the same formulas. However, this is not true if we consider an interpretation $i'$ s.t. $i'(x) = 0$ for some $x \in K$, since $m, i' \models \exists(x = 2 \land (a) \top)$ and $m', i' \models \neg(\exists(x = 2 \land (a) \top))$. 

191
It is useful in what follows to identify the set $K$ of the free clocks in a formula $\phi \in L$, defined by: $K(\bot) = \emptyset$, $K(x \triangleright r) = \{ x \}$, $K(\phi \to \psi) = K(\phi) \cup K(\psi)$, $K([a] \phi) = K(\forall \phi) = K(\phi)$, $K(\forall x. \phi) = K(\phi) \setminus \{ x \}$.

For a clock variable $y \in K$ that does not appear in the syntax of $\phi$ and $x \in K(\phi)$, we denote by $\phi\{y/x\}$ the formula obtained by uniformly substituting all the occurrences of $x$ in $\phi$ by $y$.

**Definition 5.1** Given two rational interpretations $f_-, f_+: K \to \mathbb{Q}$ and a bijection $\sigma : K \to K$, for any formula $\phi \in L$ let $\phi +_\sigma f_-/f_+$ be defined as follows, where $x \triangleright t$ for $t < 0$ should be read as $x \geq 0$:

- $\bot +_\sigma f_-/f_+ \overset{\text{def}}{=} \bot$
- $(\phi \land \psi) +_\sigma f_-/f_+ \overset{\text{def}}{=} (\phi +_\sigma f_-/f_+) \land (\psi +_\sigma f_-/f_+)$
- $(x \leq r) +_\sigma f_-/f_+ \overset{\text{def}}{=} \sigma(x) \leq (r + f_+(x))$
- $(x \geq r) +_\sigma f_-/f_+ \overset{\text{def}}{=} \sigma(x) \geq (r + f_-(x))$
- $(-\phi) +_\sigma f_-/f_+ \overset{\text{def}}{=} -(\phi +_\sigma f_-/f_-)$
- $([a] \phi) +_\sigma f_-/f_+ \overset{\text{def}}{=} [a](\phi +_\sigma f_-/f_+)$
- $(\forall x. \phi) +_\sigma f_-/f_+ \overset{\text{def}}{=} \forall \sigma(x), (\phi +_\sigma f_-[x \mapsto 0]/f_+[x \mapsto 0])$

Whenever $f_- = f_+ = f$, we write $+_\sigma f$; whenever $\sigma$ is the identity on $K$, we write $+_f f_+$. The following lemma, which can be proved by induction over the structure of formulas, and its corollaries characterize the relationships between the formulas satisfied by the same model under different interpretations.

**Lemma 5.2** Let $M = (M, \theta, \oplus)$ be a TTS, $\sigma : K \to K$, $\delta : K \to \mathbb{R}$ and $f_-, f_+: K \to \mathbb{Q}$ s.t. $f_- \leq \delta \leq f_+$. Then for any $m \in M$, $\phi \in L$ and $i \in \mathbb{N}(K)$, if $i + \delta \geq 0$ then:

$$M, m, i \models \phi \implies M, m, (i + \delta) \circ \sigma^{-1} \models \phi +_\sigma f_-/f_+.$$
Corollary 5.3 Let $M = (M, \theta, \oplus)$ be a TTS and $f : \mathcal{K} \to \mathbb{Q}$. Then for any $m \in M$, $\phi \in \mathcal{L}$ and $i \in \mathcal{S}(\mathcal{K})$, if $i + f \geq 0$ then:

$$M, m, i \models \phi \iff M, m, i + f \models \phi + f.$$  

Corollary 5.4 Let $M = (M, \theta, \oplus)$ be a TTS and $\delta : \mathcal{K} \to \mathbb{R}$. Then for any $m \in M$, $\phi \in \mathcal{L}$ and $i \in \mathcal{S}(\mathcal{K})$, if $i + \delta \geq 0$ and for any $x \in \mathcal{K}(\phi)$, $\delta(x) = 0$, then:

$$M, m, i \models \phi \iff M, m, i + \delta \models \phi.$$  

With these, we can proceed with the proof of the H-Mp. We say that a TTS $M = (M, \theta, \oplus)$ has the finite image property if for any action $a \in \Sigma$ and any $m \in M$, $\theta(m, a)$ is finite.

Theorem 5.5 (Hennessy-Milner Theorem) Consider a TTS $M = (M, \theta, \oplus)$ satisfying the finite image property. Then for any $m, n \in M$:

$$m \sim n \iff \forall i \in \mathcal{S}(\mathcal{K}), \phi \in \mathcal{L}, M, m, i \models \phi \iff M, n, i \models \phi.$$  

Proof. ($\Leftarrow$): We prove that $R$ is a bisimulation $R = \{(m, n) \mid \forall i, \forall \phi, M, m, i \models \phi \iff M, n, i \models \phi\}$.  

Supp. that $m \xrightarrow{a} m'$. If there exists no $n' \in M$ s.t. $n \xrightarrow{a} n'$, then $M, n, i \models [a]_\perp$, implying $M, m, i \models [a]_\perp$ contradicting the assumption. Let $F = \{n_k \mid n \xrightarrow{a} n_k\}$, which is finite since the TTS is image finite. Suppose that $(m', n_k) \notin R$ for any $k$.

Then, there exists $i_k \in \mathcal{S}(\mathcal{K})$ and $\phi_k \in \mathcal{L}$ s.t. $M, m', i_k \models \phi_k$ and $M, n_k, i_k \models \neg\phi_k$ for any $k$. For every $x \in \mathcal{K}(\phi_k)$, consider a new variable $x_k$ distinct from all the other variables. Let $\phi_k' = \phi_k\{x_k/\phi\}$, for every $k$ and let $i' \in \mathcal{S}(\mathcal{K})$ s.t. $i'(x_k) = i_k(x)$ for every $k$. We have: $M, m', i' \models \bigwedge_k \phi_k'$ and $M, n_k, i' \models \neg\phi_k'$ by Corollary 5.4. Then $M, m, i' \models \langle a \rangle \bigwedge_k \phi_k'$ and $M, n, i' \models [a] \bigvee \neg\phi_k'$ contradiction.

Supp. that $m' = d \oplus m$. For $r \in \mathbb{Q}_{\geq 0}$, $r \geq d$, $M, m, i[x \mapsto r - d] \models \exists \phi(x = r)$, which implies $M, n, i[x \mapsto r - d] \models \exists \phi(x = r)$. Hence, there exists $n' \in M$ s.t. $n' = d \oplus n$. We prove that $(m', n') \in R$. For any $i \in \mathcal{S}(\mathcal{K})$ and $\phi \in \mathcal{L}$, $M, m', i \models \phi$ implies, using Corollary 5.4, that $M, m', i[x \mapsto 0] \models \phi$ for any $x \notin \mathcal{K}(\phi)$; further, applying Corollary 5.3, $M, m', i[x \mapsto 0] + r \models \phi + r$. Then, $M, m, i[x \mapsto 0] + r - d \models \exists (x = r \land (\phi + r))$ and $M, n, i[x \mapsto 0] + r - d \models \exists (x = r \land (\phi + r))$. Consequently, $M, n, i[x \mapsto 0] + r - d \models \forall (x = r \rightarrow (\phi + r))$. This implies that $M, n', i[x \mapsto 0] + r \models x = r \rightarrow (\phi + [d])$ ($[d]$ denotes the smallest natural number bigger than $d$), which implies $M, n', i[x \mapsto 0] + r \models (\phi + r)$. So $M, n', i + r \models \phi + r$, and using Corollary 5.3, $M, n', i \models \phi$.

The symmetry of $R$ proves the other cases. \hfill\qed

6 Metatheory for TML

In this section we develop a proof system for TML. We prove that TML is not compact and consequently requires infinitary rules. However, we demonstrate that our proof system is strongly-complete for the TTS-semantics. The completeness proofs
of propositional logic, the axioms and the rules in Table 1. They are stated for

\[ \varepsilon \in \{a | a \in \Sigma\} \cup \{\forall x. | x \in K\}, \]

where \( \varepsilon \) is the empty word. A context is a word formed by a modal prefix \( w \in Mod^* \) concatenated with the metavariable \( X \); e.g., \( [a]X, \forall x.\varepsilon[X], \forall x.[a] \varepsilon \) are contexts. To emphasize the presence of the metavariable we will use the functional representation of type \( C \varepsilon \), \( \phi \in Mod^* \), \( \epsilon \) is the empty word. A context - the empty one - and for \( \phi \in L \), \( \varepsilon[\phi] = \phi \).

The axiomatic system of TML includes, in addition to the axioms and the rules of propositional logic, the axioms and the rules in Table 1. They are stated for arbitrary \( \phi, \psi \in L \), \( r, s, t \in Q_{\geq 0}, a \in \Sigma, \Box \in Mod \) and arbitrary context \( C[X] \).

(A1): \( \vdash x \geq 0 \)

(A2): \( \vdash (x \geq r) \lor (x \leq r) \)

(A3): \( \vdash x \leq r \rightarrow \neg(x \geq s), r < s \)

(A4): \( \vdash x \leq r \rightarrow [a](x \leq r) \)

(A5): \( \vdash \forall \phi \rightarrow \phi \land \forall \forall \phi \)

(A6): \( \vdash \forall \phi \rightarrow \forall(r \leq x \leq s \rightarrow \phi), r \leq s \)

(A7): \( \vdash x \geq r \rightarrow \forall(x \geq r) \)

(A8): \( \vdash \exists(x = r \land \phi) \rightarrow \forall(x = r \rightarrow \phi) \)

(A9): \( \vdash \exists(x \leq r \land \forall \phi) \rightarrow \forall(x \geq r \rightarrow \phi) \)

(A10): \( \vdash \exists(x \leq r) \land \exists(x \geq r) \rightarrow \exists(x = r) \)

(A11): \( \vdash x \leq r \land y \leq s \rightarrow \forall(x \leq r + t \rightarrow y \leq s + t) \)

(A12): \( \vdash \forall \exists. \varepsilon \rightarrow \phi + x \rightarrow r[x \rightarrow s], r \leq s \)

(A13): \( \vdash \forall(x \leq r).\forall(r \leq s \rightarrow \phi) \rightarrow \forall(x = r).\forall(s \leq x \leq s + r \rightarrow \phi) \)

(A14): \( \vdash \Box(\phi \rightarrow \psi) \rightarrow (\Box \phi \rightarrow \Box \psi) \)

(R1): \( \vdash \phi, \text{ then } \vdash \Box \phi \)

(R2): \( \{C[x \leq r] | r > s\} \vdash C[x \leq s] \)

(R3): \( \{C[x \geq r] | r \in Q_{\geq 0}\} \vdash C[\bot] \)

(R4): \( \{C[\phi + [x \rightarrow r][x \rightarrow s]] | r \leq s\} \vdash C[\forall x.\phi] \)

(R5): \( \{C[\forall r(x \leq s \rightarrow \phi)] | s \in Q_{\geq 0}\} \vdash C[\forall \phi] \)

The Axiomatic System of TML.

A formula \( \phi \) is provable, denoted by \( \vdash \phi \), if it can be proven from the given axioms and rules. We say that \( \phi \) is consistent, if \( \phi \rightarrow \bot \) is not provable. Given a set \( \Phi \) of formulas, we say that \( \Phi \) proves \( \phi \), \( \Phi \vdash \phi \), if from the formulas of \( \Phi \) and from the axioms one can prove \( \phi \), eventually using Boolean or infinitary-Boolean reasoning. In other words, we assume that the provability is closed under the rule
\[ \Phi \cup \{ \phi \} \vdash \psi \text{ iff } \Phi \vdash \phi \rightarrow \psi, \]
for arbitrary (possibly infinite) sets \( \Phi \subseteq \mathcal{L} \). \( \Phi \) is consistent if it is not the case that \( \Phi \vdash \bot \).

The axioms (A1)-(A3) state simple facts about the clock values. The axiom (A4) reflects the fact that action-transitions in a TTS happen instantaneously.

The axioms (A5)-(A10) describe the nature of time in TTSs. Thus, (A5) states that the time is linear and 0-delays do not perturb the system; (A6) that the time is persistent; (A7) that the flow of time is unidirectional (the past and the future are disjoint); (A8) that the time is deterministic; (A9) and (A10) that the time is continuous. The axiom (A11) guarantees that all the clocks measure the same time flow.

The axiom (A12) together with the infinitary rule (R4) describe the fact that the information provided by a clock variable \( x \) in a formula prefixed by \( \forall x \) is superfluous.

The role of (A13) is to characterize the interaction between the two types of universal quantifiers.

The axiom (A14) and the rule (R1) state that all the box-like operators of TML are normal.

The rules (R2)-(R5) are infinitary and have instances for any possible context. For instance, the formulas below are instances of the rules (R2) and (R3) respectively.

\[
\{ [a](x \geq r) \mid r < s \} \vdash [a](x \geq s),
\{ \forall [a](x \geq r) \mid r \in \mathbb{Q} \geq 0 \} \vdash \forall [a] \bot.
\]

The rule (R2) reflects the Archimedean property of rationals. (R3) guarantees that the value of any clock is finite in any model under any interpretation.

By induction on the structure of possible proofs, we prove the soundness.

**Theorem 6.1 (Soundness)** The axiomatic system in Table 1 is sound with respect to the TTS-semantics, i.e., for arbitrary \( \Phi \subseteq \phi \) and \( \phi \in \mathcal{L} \),

\[ \Phi \vdash \phi \text{ implies } \Phi \models \phi. \]

### 6.2 Non-Compactness of TML

We have seen in Section 4 that the set of validities of TML are not recursively enumerable. This means that any complete axiomatization of TML must be infinitary. Otherwise, we could enumerate all the proofs and the set of provable formulas, which in a complete logic coincides with the set of validities, is recursively enumerable - contradiction!

There is also a model theoretic result that ensures us the necessity of having infinitary rules:

**Theorem 6.2 (Non-Compactness of TML)** TML with the TTS-semantics is not compact, i.e., there exists an infinite set \( \Phi \subseteq \mathcal{L} \) such that each finite subset of \( \Phi \) admits a model but \( \Phi \) does not admit any model.
Proof. The result derives from the soundness of the infinitary rules and each rule can be used to produce examples of such sets. Consider, for example, \( s \in \mathbb{Q}_{\geq 0} \) and the set 
\[ \Phi = \{ x \geq r \mid r < s \} \cup \{ x < s \}. \]
Since for any \( i \in \mathbb{S}(K) \), \( i(x) \geq r \) for each \( r < s \) implies \( i(x) \geq s \), \( \Phi \) does not admit any model. However, it is not difficult to construct a model for just any finite subset of \( \Phi \).

6.3 Canonical Model and Completeness

In this section we prove that the axiomatic system of TML is not only sound, but also complete for the TTS-semantics, meaning that for arbitrary \( \Phi \subseteq \mathcal{L} \) and \( \phi \in \mathcal{L} \), \( \Phi \models \phi \iff \Phi \vDash \phi \). To complete this proof it is sufficient to show that any consistent formula has a model. In the following we construct a canonical model, which is a TTS such that each consistent formula is satisfied at some state under some interpretation. In modal logics such a construction is usually done using maximally consistent sets of formulas as states.

For some set \( \Lambda \subseteq \mathcal{L} \), we say that \( \Phi \subseteq \mathcal{L} \) is \( \Lambda \)-maximally consistent if \( \Phi \) is consistent and no formula of \( \Lambda \) can be added to \( \Phi \) without making it inconsistent. \( \Phi \) is maximally-consistent if it is \( \mathcal{L} \)-maximally-consistent.

The aforementioned technique to construct canonical models cannot be applied directly for TML because to the same state of a given TTS corresponds different maximally-consistent sets of formulas under different interpretations. We generalize this construction to cope with the complexity of TML. To the best of our knowledge, the following construction is original.

For the beginning, we observe that given a maximally-consistent set of formulas, the information contained about a given clock is complete.

Let \( \Omega \) be the set of \( \mathcal{L} \)-maximally consistent sets.

Lemma 6.3 For arbitrary \( \Lambda \in \Omega \) and \( x \in K \), 
\[ \sup\{ r \in \mathbb{Q}^+ \mid x \geq r \in \Lambda \} = \inf\{ r \in \mathbb{Q}^+ \mid x \leq r \in \Lambda \} \in \mathbb{R}_{\geq 0}. \]

Proof. Let \( A = \{ r \in \mathbb{Q}^+ \mid x \geq r \in \Lambda \} \) and \( B = \{ r \in \mathbb{Q}^+ \mid x \leq r \in \Lambda \} \). (A1) guarantees that \( A \neq \emptyset \) and if \( B = \emptyset \), we can derive a contradiction from (R3) for \( C[X] = X \).

Since the two sets are non-empty, the sup and inf exist. Moreover, (R3) can also be used to prove that \( \sup A < \infty \). Let \( \sup A = u \) and \( \inf B = v \). If \( u < v \), there exists \( r \in \mathbb{Q}^+ \) such that \( u < r < v \). Hence, \( x \leq r \in \Lambda \), which contradicts \( r \leq v \). If \( v < u \), there exists \( r_1, r_2 \in \mathbb{Q}^+ \) such that \( v < r_1 < r_2 < u \). Hence, \( x \leq r_i, x \geq r_i \in \Lambda \) for \( i = 1, 2 \). Since \( r_2 - r_1 > 0 \), (A3)\( \vdash x \geq r_2 \rightarrow \neg(x \leq r_1) \), which proves the inconsistency of \( \Lambda \) - contradiction. \( \square \)
The previous lemma demonstrates that to each maximally-consistent set corresponds a unique interpretation of clocks that we will identify in what follows using the function \( \mathcal{I} : \Omega \rightarrow \mathcal{I}(K) \) defined for arbitrary \( \Lambda \in \Omega \) and \( x \in K \) by
\[
\mathcal{I}(\Lambda)(x) = \sup\{ r \in \mathbb{Q}^+ \mid x \geq r \in \Lambda \} \in \mathbb{R}_{\geq 0}.
\]
Since \( \mathcal{I}(\Lambda) \) synthesize only the information in \( \Lambda \) regarding the clocks, there exist disjoint sets \( \Lambda_1, \Lambda_2 \in \Omega \) s.t. \( \mathcal{I}(\Lambda_1) = \mathcal{I}(\Lambda_2) \); this equality defines an equivalence relation on \( \Omega \) and the equivalence classes are in one to one correspondence with the interpretations in \( \mathcal{I}(K) \).

Observe that any state in a model of TML corresponds to a function from \( \mathcal{I}(K) \) to \( \Omega \): given a model, each interpretation identifies a maximally-consistent set of formulas satisfied by that model under the given interpretation. Consequently, to construct the canonical model we will have to take as states not maximally-consistent sets of formulas, but functions from interpretations to maximally-consistent sets. However, not just any function \( \gamma : \mathcal{I}(K) \rightarrow \Omega \) is a good candidate for a model, because between the maximally-consistent sets associated to a model under different interpretation there are certain coherence conditions as the ones described in Lemma 5.2 and corollaries 5.3 and 5.4. These coherences are formally described in the next definition.

**Definition 6.4** A function \( \gamma : \mathcal{I}(K) \rightarrow \Omega \) is **coherent**, if for any \( i \in \mathcal{I}(K) \), any bijection \( \sigma \) on \( K \), any \( \delta : K \rightarrow \mathbb{R} \) s.t. \( i + \delta \geq 0 \), and any \( f_-, f_+ : K \rightarrow \mathbb{Q} \) s.t. \( f_- \leq \delta \leq f_+ \),
1. \( (\mathcal{I} \circ \gamma)(i) = i \);
2. if \( \phi \in \gamma(i) \), then \( \phi +_{\sigma} f_-/f_+ \in \gamma((i + \delta) \circ \sigma^{-1}) \).

The first fundamental result that we prove is that any maximally-consistent set \( \Lambda \) belongs to the image \( \gamma(\mathcal{I}(K)) \) of some coherent function \( \gamma \). After constructing the canonical model on the set of coherent functions, this result will guarantee that any maximally-consistent set is satisfied by some model under some interpretation.

**Lemma 6.5** For any \( \Lambda \in \Omega \), there exists a coherent function \( \gamma \) s.t. \( \gamma(\mathcal{I}(\Lambda)) = \Lambda \).

**Proof.** We say that a set \( S \subseteq \Omega \) is **coherent** if there exists a set \( T \subseteq \mathcal{I}(K) \) and a bijection \( \gamma : T \rightarrow S \) that satisfies the two conditions of Definition 6.4. \( S \) is **maximally-coherent** if \( T = \mathcal{I}(K) \). Observe that the bijection that defines a maximally-coherent set is a coherent function in the sense of Definition 6.4.

We prove, using a transfinite induction, that any coherent set has a maximally-coherent extension. And this proves the existence of \( \gamma \), because \( \{ \Lambda \} \) is coherent.

**I.** Firstly, observe that \( S \subseteq \Omega \) is coherent iff for any \( \Lambda_1, \Lambda_2 \in \Omega \), with \( i_s = \mathcal{I}(\Lambda_s) \), \( s = 1, 2 \),
\[
\Lambda_1 + (i_2 - i_1) \subseteq \Lambda_2 \quad \text{and} \quad \Lambda_2 + (i_1 - i_2) \subseteq \Lambda_1,
\]
where for arbitrary \( \Lambda' \in \Omega \) and \( \delta : K \rightarrow \mathbb{R} \),
\[
\Lambda' + \delta = \{ \phi +_{f_-/f_+} f_- \mid f_-, f_+ : K \rightarrow \mathbb{Q}, f_- \leq \delta \leq f_+ \}.
\]
Moreover, \( \Lambda_1 + (i_2 - i_1) \subseteq \Lambda_2 \) iff \( \Lambda_2 + (i_1 - i_2) \subseteq \Lambda_1 \).
II. If \( i = \mathcal{I}(\Lambda) \) and \( i' \in \mathcal{I}(K) \), then there exists \( \Lambda' \in \Omega \) s.t. \( \mathcal{I}(\Lambda') = i' \) and \( \{\Lambda, \Lambda'\} \) is coherent. To prove this, we firstly use Definition 5.1 and the axioms to prove that \( \Lambda + (i' - i) \) is consistent. Then, we observe that all the infinitary rules have countable sets of instances, which allows us to apply Rasiowa-Sikorski lemma to conclude that \( \Lambda + (i' - i) \) must have a maximal-consistent extension \( \Lambda' \). Since \( \Lambda + (i' - i) \subseteq \Lambda' \), we also have \( \Lambda' + (i' - i) \subseteq \Lambda \). Hence, \( \{\Lambda, \Lambda'\} \) is coherent.

III. If \( S = \{\Lambda_0, \Lambda_1, \ldots, \Lambda_k\} \) is a finite coherent set, \( i_s = \mathcal{I}(\Lambda_s), s = \overline{1..k} \) and \( j \in \mathcal{I}(K) \), then there exists \( \Lambda_j \in \Omega \) s.t. \( \mathcal{I}(\Lambda_j) = j \) and \( S \cup \{\Lambda_j\} \) is coherent. Supp. \( \Lambda = \Lambda_0 \). It is sufficient to show that the following set is consistent, which is not a difficult task:

\[
X = \bigcup_{s \in \overline{1..k}} (\Lambda_s + (j - i_s)).
\]

IV. If \( S = \{\Lambda_s \mid s \in \mathbb{N}\} \) is a countable coherent set (supp. \( \Lambda = \Lambda_0 \)) and \( j \in \mathcal{I}(K) \). There exists \( \Lambda_j \in \Omega \) s.t. \( S \cup \{\Lambda_j\} \) is coherent. As before, we need to prove that

\[
X = \bigcup_{s \in \mathbb{N}} (\Lambda_s + (j - i_s))
\]

is consistent. Let \( X_t = \bigcup_{s \leq t} (\Lambda_s + (j - i_s)) \) for \( t \in \mathbb{N} \). We have previously proved that \( X_t \) is consistent for any \( t \in \mathbb{N} \). Moreover, observe that \( X = \bigcup_{t \in \mathbb{N}} X_t \).

Suppose that \( X \) is not consistent. If the contradiction can be proven without involving the infinitary rules, then there must exist one \( X_t \) that contains all the formulas in the proof, which contradicts the consistency of \( X_t \). Consequently, if \( X \) is inconsistent it is because, when \( t \) goes to infinity, the sets \( X_t \setminus X_{t-1} \) collect the left-hand side of some of our infinitary rules.

If for arbitrary contexts \( C \), formulas \( \phi \in \mathcal{L} \), \( r \in \mathbb{Q}_{\geq 0} \) and \( x \in \mathcal{K} \) we denote by

\[
\mathcal{L}_1(C, x) = \{C[x \leq s], C[x \geq s] \mid s \in \mathbb{Q}_{\geq 0}\}
\]

\[
\mathcal{L}_2(C, x, \phi) = \{C[\phi + [x \rightarrow s]/[x \rightarrow s']] \mid s, s' \in \mathbb{Q}, s \leq s'\},
\]

\[
\mathcal{L}_3(C, x, \phi, r) = \{C[\forall (r \leq x \leq s \rightarrow \phi)] \mid r, s \in \mathbb{Q}_{\geq 0}, r \leq s\},
\]

one can prove that for each \( t \in \mathbb{N} \), \( X_t \) is \( \mathcal{L}_1(C, x) \)-maximally consistent and for each \( L \) of type \( \mathcal{L}_2(C, x, \phi) \) or \( \mathcal{L}_3(C, x, \phi, r) \), either \( X_t \cap L = \emptyset \) or \( X_t \) is \( L \)-maximally consistent. This demonstrates that if in \( X \) we can use some infinitary rule, it must be used within some \( X_t \) for some \( t \in \mathbb{N} \). Consequently, the consistency of \( X_t \) proves the consistency of \( X \).

V. If \( S \subseteq \Omega \) is an uncountable coherent set and \( j \in \mathcal{I}(K) \), then there exists \( \Lambda_j \in \Omega \) s.t. \( S \cup \{j\} \) is coherent. Supp. \( S = \{\Lambda_s \mid s \in \mathbb{R}\} \), \( \Lambda = \Lambda_0 \) and \( i_s = \mathcal{I}(\Lambda_s) \). As in the other cases it is sufficient to prove that the following set is consistent:

\[
Y = \bigcup_{s \in \mathbb{R}} (\Lambda_s + (j - i_s)).
\]

Since \( Y \subseteq \mathcal{L} \) and \( \mathcal{L} \) is countable, \( Y \) is countable and consequently we can assume that

\[
Y = \bigcup_{s \in \mathbb{N}} (\Lambda_s + (j - i_s)).
\]

Further the result is a consequence of IV. \( \square \)
In what follows we define a TTS using the set
\[ \Gamma = \{ \gamma : \mathcal{I}(K) \to \Omega \mid \gamma \text{ is a coherent function} \} \]
as the support-set of the model and the structure

- \( \gamma \overset{\phi}{\rightarrow} \gamma' \) if \([\forall i \in \mathcal{I}(K), [a] \phi \in \gamma(i) \Rightarrow \phi \in \gamma'(i)]\);
- \( \phi' = d \oplus \gamma \) if \([\forall i, \forall \phi \in \gamma(i) \Rightarrow \phi \in \gamma'(i + d)]\).

**Theorem 6.6 (Canonical Model)** The tuple \( (\Gamma, \theta, \oplus) \) defined above is a TTS.

**Proof.** Firstly, we prove that \( \gamma_1 = d \oplus \gamma \) and \( \gamma_2 = d \oplus \gamma \) implies \( \gamma_1 = \gamma_2 \).

For any \( \forall i \in \mathcal{I}(K) \) and \( x \in K \setminus K(\phi), \phi \in \gamma_1(i) \) implies \( x = [d] \wedge \phi \in \gamma_1(i[x \mapsto [d]]) \), where \([d] \) denote the smallest integer larger than \( d \). Since \( \gamma_1 = d \oplus \gamma \),
\[ \exists (x = [d] \wedge \phi) \in \gamma_1(i[x \mapsto [d]] - d), \]
which further implies, due to (A8), that
\[ \forall (x = [d] \rightarrow \phi) \in \gamma_1(i[x \mapsto [d]] - d). \]
\( x = [d] \rightarrow \phi \in \gamma_2(i[x \mapsto [d]]) \) since \( \gamma_2 = d \oplus \gamma \). Then \( \phi \in \gamma_2(i[x \mapsto [d]]) \), which implies \( \phi \in \gamma_2(i) \) because \( x \not\in K(\phi) \). Hence for any \( i \in \mathcal{I}(K), \gamma_1(i) \subseteq \gamma_2(i) \).

Similarly, \( \gamma_2(i) \subseteq \gamma_2(i) \).

Observe now that due to (A5), \( \forall \phi \in \gamma(i) \) implies \( \phi \in \gamma(i) \), for any \( \gamma \in \Gamma \) and \( i \in \mathcal{I}(K) \). Hence, \( 0 \oplus \gamma = \gamma \).

Now we prove that \( d \oplus (d' \oplus m) \overset{\sim}{=} (d + d') \oplus m \), i.e. \( \exists \gamma_1, \gamma_2 \) s.t. \( \gamma_1 = d' \oplus \gamma, \gamma_2 = d \oplus \gamma_1 \) iff \( \exists \gamma_3 \) s.t. \( \gamma_3 = (d + d') \oplus \gamma \).

(\( \Rightarrow \)) Suppose \( \exists \gamma_1, \gamma_2 \) s.t. \( \gamma_1 = d \oplus \gamma, \gamma_2 = d' \oplus \gamma_1 \). From (A5), \( \forall \phi \in \gamma(i) \) implies \( \forall \phi \in \gamma(i) \) and since \( \gamma_1 = d \oplus \gamma, \forall \phi \in \gamma_1(i + d) \). This implies \( \phi \in \gamma_2(i + d + d') \) because \( \gamma_2 = d' \oplus \gamma_1 \). Hence, there exists \( \gamma_3 = (d + d') \oplus \gamma \) and, in fact, \( \gamma_3 = \gamma_2 \).

(\( \Leftarrow \)) Supp. that \( \exists \gamma_3 \) s.t. \( \gamma_3 = (d + d') \oplus \gamma \).

Firstly, we prove that there exists \( \gamma_1 \) s.t. \( \gamma_1 = d' \oplus \gamma \) by constructing it. For each \( i \in \mathcal{I}(K), \gamma(i) + \sigma d' \subseteq \gamma_1(i + d'). \) It is not difficult to verify that for arbitrary \( i, j \in \mathcal{I}(K), \gamma(i) + \sigma d' + (j - i) \subseteq \gamma(j) + \sigma d' \) and
\[ \bigcup_{r \in \mathcal{K}} \{ x \leq r \mid r \in \mathbb{Q}^+, i(x) \leq r \} \subseteq \gamma(i) + \sigma d'. \]
Now, we can use a similar transfinite construction as in Lemma 6.5 to construct entire \( \gamma_1 \).

Secondly, we need to prove that there exists \( d \oplus \gamma_1 \). For any \( i \in \mathcal{I}(K), \forall \phi \in \gamma_1(i) \) implies \( x \leq r \wedge \forall \phi \in \gamma_1(i) \) for \( r \in \mathbb{Q}^+ \) s.t. \( i(x) \leq r \leq i(x) + d \). Then,
\[ x \leq r + [d'] \wedge \forall \phi \in [d'] \in \gamma_1(i + [d']). \]
Now because \( \gamma_1 = d' \oplus \gamma \), we obtain
\[ \exists (x \leq r + [d'] \wedge \forall \phi \in [d']) \in \gamma_1(i + [d'] - d'). \]
From this we get, by applying (A9), that
\[ \forall (x \geq r + [d'] \rightarrow \forall \phi \in [d']) \in \gamma_1(i + [d'] - d'). \]
Since \( \gamma_3 = (d + d') \oplus \gamma \),
\[ x \geq r + [d'] \rightarrow \forall \phi \in [d'] \in \gamma_3(i + [d'] + d). \]
And because \( r \leq i(x) + d, x \geq r + [d'] \in \gamma_3(i + [d'] + d) \). We then get that
∀(ϕ + [d']) ∈ γ₃(i + [d'] + d), implying W ϕ ∈ γ₃(i + d), which using (A5) gives ϕ ∈ γ₃(i + d). Hence, there exists d ⊕ γ₁ = γ₃.

Lemma 6.7 (Truth Lemma) For any ψ ∈ L, i ∈ ℐ(Κ) and γ ∈ Γ,

Γ, γ, i |= ψ iff ψ ∈ γ(i).

Proof. Induction on ψ.

[The case ψ = ∀x.ϕ]: Γ, γ, i |= ∀x.ϕ iff Γ, γ, i[x → u] |= ϕ for any u ∈ ℜ≥, i.e., ϕ ∈ γ(i[x → u]) for all u ∈ ℜ≥.
(⇒) ϕ ∈ γ(i[x → u]) implies ϕ + [x → r]/[x → s] ∈ γ(i) for all r, s ∈ ℜ s.t. r ≤ u − i(x) ≤ s. This must happen for any u ∈ ℜ≥, so for any r, s ∈ ℜ, ϕ + [x → r]/[x → s] ∈ γ(i). Hence, ∀x.ϕ ∈ γ(i) by rule (R4).
(⇐) Using (A12), ∀x.ϕ ∈ γ(i) implies ϕ + [x → r]/[x → s] ∈ γ(i[x → u]) for any u ∈ ℜ≥ and r, s ∈ ℜ s.t. r ≤ u − i(x) ≤ s. Consequently, ∀x.ϕ ∈ γ(i[x → u]) by (R4). And (A12), ϕ ∈ γ(i[x → u]) for all u ∈ ℜ≥.

[The case ψ = Wϕ]: Γ, γ, i |= Wϕ iff for any d ∈ ℜ≥ and γ′ ∈ Γ s.t. γ′ = d ⊕ γ, Γ, γ′, i + d |= ϕ if ϕ ∈ γ′(i + d) by the inductive hypothesis.
(⇒) ϕ ∈ γ′(i + d) implies (x = r) ∧ ϕ ∈ γ′((i + d)[x → r]), for any r ∈ ℚ⁺, r ≥ d. Applying (A8), W(x = r → ϕ) ∈ γ′(i[i[x → r − d] → r]) for any r ∈ ℚ⁺, r ≥ d. This implies x ≤ r → W(x = r → ϕ) ∈ γ′(i[x → u]) for any u ∈ ℜ≥. Then, ∀(x ≤ r). W(x = r → ϕ) ∈ γ(i). Applying (A13), ∀(x = r). W(r ≤ x ≤ 2r → ϕ) ∈ γ(i), which further implies ∀(x = 0). W(0 ≤ x ≤ r → ϕ) ∈ γ(i) by (A12) and (R4). Using (R5), ∀(x = 0). W(x ≥ 0 → ϕ) ∈ γ(i), which implies ∀(x = 0). W(x ≥ 0 → ϕ) ∈ γ(i[x → 0]). So, x = 0 → W(x ≥ 0 → ϕ) ∈ γ(i[x → 0]), which implies W(x ≥ 0 → ϕ) ∈ γ(i[x → 0]). Using (A1), Wϕ ∈ γ(i[x → 0]) and further, Wϕ ∈ γ(i).
(⇐) derives from the definition of ⊕.

[The case ψ = [a]ϕ]: Γ, γ, i |= [a]ϕ iff for any γ′ ∈ Γ s.t. γ → γ′, Γ, γ′, i |= ϕ, iff ϕ ∈ γ′(i) by induction.
(⇒) Supp. (a)ϕ ∈ γ(i). Let A = {γ′ | γ → γ′}, B = {ϕ} ∪ {ψ | [a]ψ ∈ γ(i)} ∪ Δ, and B_j = {ψ | [a]ψ ∈ γ(j)} ∪ Δ, for any j ≠ i, where Δ = ∪_{x ∈ K} {x ≥ r | r ≥ i(x)}. \{ψ | [a]ψ ∈ γ(i)} ∪ Δ and B_j, j ≠ i are consistent.
Suppose that B_j is inconsistent. Then, there exists a set F ⊆ B_j s.t. F ⊥ ϕ. If F is finite, (R1) guarantees that [a]F ⊥ [a]ϕ, where [a]F = \{[a]ρ | ρ ∈ F\}. Otherwise, F ⊥ ϕ is (modulo Boolean reasoning possible involving infinite meets) an instance of one of the rules (R2)-(R5); in all these cases, [a]F ⊥ [a]ϕ is an instance of the same rule for the context C[X] = [a]X. Since F ⊆ B_j, [a]F ⊆ γ(i) implying [a]ϕ ∈ γ(i), which contradicts the consistency of γ(i). Hence, B_j is consistent.
Now we prove that for any j, j′ ∈ ℐ(Κ), B_j and B_j′ are such that B_j + (j′ − j) ⊆ B_j′. If j ≠ i, then for arbitrary ρ ∈ B_j either [a]ρ ∈ γ(j), or ρ = x ≤ r. In the first case, [a]ρ +_s f → f_s ∈ γ(j′), for all f_s ≤ j′ − j ≤ f_s. So, ρ +_s f → f_s ∈ B_j′. In the second case, since ρ = x ≤ r is closed under any interpretation transformation, for
any $f_- \leq j' - j \leq f_+$, $\rho +_{\sigma} f_-/f_+ \in B_{j'}$.
If $j = i$, consider an arbitrary $\rho \in B_j$. If $\rho \neq \neg \phi$, we get a similar case as above.
Otherwise, $\langle a \rangle \rho \in \gamma(i)$, which implies $\langle a \rangle \rho +_{\sigma} f_-/f_+ \in \gamma(j)$ for all $f_- \leq j - i \leq f_+$.
So, $\rho +_{\sigma} f_-/f_+ \in B_j$.

At this point we can use a similar strategy as in Theorem 6.5 to prove that there exists $\gamma'' \in \Gamma$ s.t. for any $j \in \mathcal{I}(\Lambda)$, $B_j \subseteq \gamma''(j)$. But then, $\gamma'' \in \Delta$, which implies $\phi \in \gamma''(i)$ - contradiction!
Hence, $[a] \phi \in \gamma(i)$.

Thus, $\Phi \subseteq \mathcal{L}$ is consistent, there exists $\gamma \in \Gamma$ s.t.

\begin{equation*}
\text{Corollary 6.8: If } \Phi \subseteq \mathcal{L} \text{ is consistent, there exists } \gamma \in \Gamma \text{ s.t. } \Gamma, \gamma, i \models \Phi.
\end{equation*}

Proof. Because the infinitary rules of TML have countable sets of instances, the Rasiowa-Sikorski lemma guarantees the existence of some $\Lambda \in \Omega$ s.t. $\Lambda \supseteq \Phi$. Lemma 6.5 guarantees that there exists $\gamma \in \Gamma$ s.t. $\Lambda = \gamma(\mathcal{I}(\Lambda))$. Hence $\Gamma, \gamma, \mathcal{I}(\Lambda) \models \Phi$ by applying Lemma 6.7.

7 Conclusions

In this paper we addressed and solved a series of open problems regarding the timed logics and real-time systems. We develop the metatheory for the Timed Modal Logic (TML), which is the most basic logic used in practice for specifying and analyzing timed transition systems (TTSs).

In this paper we demonstrate that TML is adequate for the TTS semantics. We show that its satisfiability problem is undecidable and the set of the validities is not recursively enumerable. Despite this, we develop a strongly-complete proof system for TML. Because TML is not compact, the proof system contains necessarily infinitary rules and these rules also explain why the set of validities is not recursively enumerable. Our axioms characterize the concept of time used in the definition of TTS. Our completeness proof is based on a novel method that generalizes the classic filtration technique used in modal logics for the construction of canonical models. Essential in the proof is the use of the Rasiowa-Sikorski lemma.

All these results open new perspective on real-time systems and on their analysis and reveal new research directions. The proof system contains similar axioms to those
of other well-known logics, which makes one think about other possible connections and research perspectives.

References


Appendix

This appendix contains the details of the proofs of the major results presented in the paper.

Proof. [Proof of Theorem 4.2] Our reduction is similar to the one in [3]. The question as to whether a non-deterministic 2-counter machine has a computation with the initial location being visited infinitely often is $\Sigma_1$-hard. We show how to reduce this problem into a TML satisfiability question.

Let $M$ be a 2-counter machine with counters $X$ and $Y$ and with $n + 1$ program instructions $\ell_0, \ldots, \ell_n$. The instruction $\ell_n$ represents termination, and each instruction $\ell_i$ ($i < n$) is either an increment of the form $[\ell_i : X := X + 1; \text{goto } \ell_j]$ or a decrement of the form $[\ell_i : \text{if } X \neq 0 \text{ then } X := X - 1; \text{goto } \ell_j \text{ else goto } \ell_k]$ or a non-deterministic jump of the form $[\ell_i : \text{goto } \ell_j \text{ or } \ell_k]$. A configuration of $M$ is a triple $\langle \ell_i, x, y \rangle$, where $x$ and $y$ are natural numbers representing the current values of $X$ and $Y$. A computation of $M$ is a “valid” sequence of configurations starting in $\langle \ell_0, 0, 0 \rangle$ and ending in some configuration of the type $\langle \ell_n, x, y \rangle$.

We encode the computation of $M$ in TML using the actions $\Sigma = \{\ell_0, \ldots, \ell_n, X, Y\}$. We say that a state $m$ of a given TTS $T$ encodes the configuration $\langle \ell_i, x, y \rangle$ in the interval $(a, b)$, with $a, b \in \mathbb{R}^+$ and $a < b$, iff the following holds:

- $(d \oplus m) X \rightarrow$ for exactly $x$ distinct time-points $d$ in $(a, b)$;
- $(d \oplus m) Y \rightarrow$ for exactly $y$ distinct time-points $d$ in $(a, b)$;
- $(d \oplus m) \ell_i \rightarrow$ for $d = a$;
- $(d \oplus m) \ell_j \not\rightarrow$ for all time-points $d$ in $(a, b)$ and for all $j$.

Let $\{\langle \ell_{ij}, x_j, y_j \rangle : j \geq 0\}$ be a computation of $M$ (i.e. $i(0) = 0$). We may construct a (closed) TML formula $\phi^M$, such that for any TTS $T$ and any state $m$, $T, m \models \phi$ if and only if $m$ encodes the $j$’th configuration over the interval $(j, j + 1)$ for all $j \geq 0$. Also $\phi^M$ will ensure that $m$ encodes a computation where $\ell_0$ is visited infinitely often. The formula $\phi^M$ is obtained as the conjunction of a formula expressing the initial configuration, a formula expressing infinite repetition of the initial location and a formula for each instruction of $M$ ensuring that states $(d \oplus m)$ of $T$ being separated by a delay of 1 correctly encodes the given instruction. The following formula ensures that instructions of $M$ are unique:

- $\forall i (\langle \ell_i \rangle \rightarrow [\ell_j] \bot)$, for all $i \neq j$ (uniqueness);

The initial configuration may be encoded as the conjunction of the following TML formulae:

- $\langle \ell_0 \rangle \top$;
- $x$ in $\mathbb{W}(0 < x < 1 \rightarrow [\ell_i] \bot)$, for all $i \geq 0$;
(iii) \( x \) in \( \mathbb{W}(0 < x < 1 \rightarrow ([X] \perp [Y] \perp)) \).

An increment statement of the form \([\ell_i : X := X + 1; \text{goto } \ell_j]\) is reflected by the conjunction of the following formulae relating the behaviour in a unit-interval \( \{(d \oplus m) : d \in [j, j + 1]\} \) with \((j \oplus m) \xrightarrow{\ell_i} \) with the successor unit-interval:

- (i) \( \mathbb{W}((\ell_i) \top \rightarrow x \in \exists!(x = 1 \land (\ell_j)) \top) \);
- (ii) \( \mathbb{W}((\ell_i) \top \rightarrow x \in \mathbb{W}(0 < x < 1 \rightarrow [\ell_j] \perp), \text{ for all } i, j \);
- (iii) \( \mathbb{W}((\ell_i) \top \rightarrow x \in \mathbb{W}(x < 1 \land (X) \top \rightarrow y \in \exists!(y = 1 \land (X) \top)) \);
- (iv) \( \mathbb{W}((\ell_i) \top \rightarrow x \in \mathbb{W}(x < 1 \land (X) \top \land y \in \mathbb{W}(y > 0 \land x < 1 \rightarrow [X] \perp)) \rightarrow (y \in (\exists!(y > 1 \land x < 2 \land (X) \top)) \land y \in (\mathbb{W}(y > 1 \land x < 2 \land (X) \top \rightarrow z \in \mathbb{W}(z > 0 \land x < 2 \rightarrow [X] \perp)))))) \)

Here (1) encodes the goto of the instruction. (2) ensures that \( \ell_i \) actions are only possible at integer-points. (3) ensures that all \( X \) actions in the interval \([j, j + 1]\) are copied to the successor interval \([j + 1, j + 2]\). The most involved formula (4) ensures that exactly one additional \( X \) action is inserted in \([j + 1, j + 2]\) after the copy of the last \( X \) action in \([j, j + 1]\). The formulae for decrement and non-deterministic choice are similar (and simpler). Infinite repetition of \( \ell_0 \) is easily expressed as \( \mathbb{W}((\ell_0) \rightarrow x \in \exists!(x > 0 \land (\ell_0) \top)) \).

**Proof.** [Proof of Lemma 5.2] Induction on \( \phi \).

**[The case \( x \geq r \):]** \( (x \geq r) +_{\sigma} f^-/f_+ = \sigma(x) \geq (r + f_-(x)) \). \( M, m, i \models x \geq r \) implies \( i(x) \geq r \). Then \((i(\delta) \circ \sigma^{-1})(x) \geq r + \delta(x) \geq r + f_-(x) \). Hence \( M, m, (i(\delta) \circ \sigma^{-1}) \models (x \geq r) +_{\sigma} f^-/f_+ \).

**[The case \( x > r \):]** \( (x > r) +_{\sigma} f^-/f_+ = \sigma(x) > (r + f_-(x)) \). \( M, m, i \models x > r \) implies \( i(x) > r \). Then \((i(\delta) \circ \sigma^{-1})(x) > r + \delta(x) \geq r + f_-(x) \). Hence \( M, m, (i(\delta) \circ \sigma^{-1}) \models (x > r) +_{\sigma} f^-/f_+ \).

**[The case \( x \leq r \):]** \( (x \leq r) +_{\sigma} f^-/f_+ = \sigma(x) \leq (r + f_+(x)) \). \( M, m, i \models x \leq r \) implies \( i(x) \leq r \). Then \((i(\delta) \circ \sigma^{-1})(x) \leq r + \delta(x) \leq r + f_+(x) \). Hence \( M, m, (i(\delta) \circ \sigma^{-1}) \models (x \leq r) +_{\sigma} f^-/f_+ \).

**[The case \( x < r \):]** \( (x < r) +_{\sigma} f^-/f_+ = \sigma(x) < (r + f_+(x)) \). \( M, m, i \models x < r \) implies \( i(x) < r \). Then \((i(\delta) \circ \sigma^{-1})(x) < r + \delta(x) \geq r + f_+(x) \). Hence \( M, m, (i(\delta) \circ \sigma^{-1}) \models (x < r) +_{\sigma} f^-/f_+ \).

**[The case \( \phi \land \psi \):]** \( M, m, i \models \phi \land \psi \) implies \( M, m, i \models \phi \) and \( M, m, i \models \psi \). By inductive hypothesis, \( M, m, (i(\delta)) \circ \sigma^{-1} \models (\phi +_{\sigma} f^-/f_+) \) and \( M, m, (i(\delta)) \circ \sigma^{-1} \models (\psi +_{\sigma} f^-/f_+) \), which imply \( M, m, i(\delta) \models (\phi +_{\sigma} f^-/f_+) \land (\psi +_{\sigma} f^-/f_+) \). Hence \( M, m, (i(\delta)) \circ \sigma^{-1} \models (\phi \land \psi) +_{\sigma} f^-/f_+ \).

204
[The case $\phi \lor \psi$]: $M, m, i \models \phi \lor \psi$ implies $M, m, i \models \phi$ or $M, m, i \models \psi$. By inductive hypothesis, $M, m, (i + \delta) \circ \sigma^{-1} \models \phi +_{\sigma} f_{-}/f_{+}$ or $M, m, (i + \delta) \circ \sigma^{-1} \models \psi +_{\sigma} f_{-}/f_{+}$, which imply $M, m, (i + \delta) \circ \sigma^{-1} \models (\phi +_{\sigma} f_{-}/f_{+}) \lor (\psi +_{\sigma} f_{-}/f_{+})$. Hence $M, m, (i + \delta) \circ \sigma^{-1} \models (\phi \lor \psi) +_{\sigma} f_{-}/f_{+}$.

[The case $[a] \phi$]: $M, m, i \models [a] \phi$ implies for any $m' \in M$ s.t. $m \xrightarrow{a} m'$, $M', m', i \models \phi$. By inductive hypothesis, $M', m', (i + \delta) \circ \sigma^{-1} \models \phi +_{\sigma} f_{-}/f_{+}$, which implies $M, m, (i + \delta) \circ \sigma^{-1} \models [a] \phi +_{\sigma} f_{-}/f_{+}$. Hence $M, m, (i + \delta) \circ \sigma^{-1} \models ([a] \phi) +_{\sigma} f_{-}/f_{+}$.

[The case $\langle a \rangle \phi$]: $M, m, i \models \langle a \rangle \phi$ implies exists $m' \in M$ s.t. $m \xrightarrow{a} m'$ and $M', m', i \models \phi$. By inductive hypothesis, $M', m', (i + \delta) \circ \sigma^{-1} \models \phi +_{\sigma} f_{-}/f_{+}$, which implies $M, m, (i + \delta) \circ \sigma^{-1} \models \langle a \rangle \phi +_{\sigma} f_{-}/f_{+}$. Hence $M, m, (i + \delta) \circ \sigma^{-1} \models (\langle a \rangle \phi) +_{\sigma} f_{-}/f_{+}$.

[The case $\forall x. \phi$]: $M, m, i \models \forall x. \phi$ implies for any $t \in \mathbb{R}_+$, $M, m, i[x \mapsto t] \models \phi$. By inductive hypothesis, $M, m', (i[x \mapsto t] + \delta[x \mapsto 0]) \circ \sigma^{-1} \models \phi +_{\sigma} f_{-}[-x \mapsto 0]/f_{+}[x \mapsto 0]$, which implies $M, m', ((i + \delta) \circ \sigma^{-1})[x \mapsto t] \models \phi +_{\sigma} f_{-}[-x \mapsto 0]/f_{+}[x \mapsto 0]$. Then $M, m, (i + \delta) \circ \sigma^{-1} \models \forall x. (\phi +_{\sigma} f_{-}[-x \mapsto 0]/f_{+}[x \mapsto 0])$. Hence $M, m, (i + \delta) \circ \sigma^{-1} \models (\forall x. \phi) +_{\sigma} f_{-}/f_{+}$.

[The case $\exists x. \phi$]: $M, m, i \models \exists x. \phi$ implies exists $t \in \mathbb{R}_+$ s.t. $M, m, i[x \mapsto t] \models \phi$. By inductive hypothesis, $M, m', (i[x \mapsto t] + \delta[x \mapsto 0]) \circ \sigma^{-1} \models \phi +_{\sigma} f_{-}[-x \mapsto 0]/f_{+}[x \mapsto 0]$. So $M, m', ((i + \delta) \circ \sigma^{-1})[x \mapsto t] \models \phi +_{\sigma} f_{-}[-x \mapsto 0]/f_{+}[x \mapsto 0]$. Then $M, m, (i + \delta) \circ \sigma^{-1} \models \exists x. (\phi +_{\sigma} f_{-}[-x \mapsto 0]/f_{+}[x \mapsto 0])$. Hence $M, m, (i + \delta) \circ \sigma^{-1} \models (\exists x. \phi) +_{\sigma} f_{-}/f_{+}$.

Proof. [Proof of Corollary 5.3] $\Rightarrow$: from Lemma 5.2.
$(\Leftarrow)$: Since $i + f - f \geq 0$, $M, m, i + f \models \phi + f$ implies $M, m, i + f - f \models \phi + f - f$ due to Lemma 5.2.

Proof. [Proof of Corollary 5.4] Let $f_{-}, f_{+} : \mathcal{K} \rightarrow \mathbb{Q}$ be such that $f_{-} \leq \delta \leq f_{+}$ and $f_{-}(x) = f_{+}(x) = 0$ for any $x \in \mathcal{K} \phi$. Then, $\phi + f_{-}/f_{+} = \phi - f_{+}/f_{-} = \phi$.
$(\Rightarrow)$: derives from Lemma 5.2.
$(\Leftarrow)$: Since $i + \delta - \delta \geq 0$, $M, m, i + \delta \models \phi$ implies $M, m, i + \delta - \delta \models \phi$ using Lemma 5.2.

Proof. [Proof of Theorem 5.5] $\Rightarrow$: Induction on $\phi$.

[The Case $[a] \phi$]: $M, m, i \models [a] \phi$ implies for any $m' \in M$ s.t. $m \xrightarrow{a} m'$, $M', m', i \models \phi$. Since $m \sim n$, so exists $n' \in M$ s.t. $n \xrightarrow{a} n'$ and $m' \sim n'$. By inductive hypothesis, $M', m', i \models \phi$ implies $M', n', i \models \phi$. Hence $M, m, i \models [a] \phi$ implies $M, n, i \models [a] \phi$. Similarly $M, n, i \models [a] \phi$ implies $M, m, i \models [a] \phi$. 205
Suppose $\Lambda \in \mathbb{R}_{\geq 0}$ and $m' \in M$ s.t. $m' = d \oplus m$, $M, m', i \models \phi$. Since $m \sim n$, so exists $n' \in M$ s.t. $n' = d \oplus n$ and $m' \sim n'$. By inductive hypothesis, $M, m', i \models \phi$ implies $M, n', i \models \phi$. Hence $M, m, i \models \forall \phi$ implies $M, n, i \models \forall \phi$. Similarly $M, n, i \models \forall \phi$ implies $M, m, i \models \forall \phi$.

**Proof.** [Proof of Lemma 6.3] First, the sets $A = \{r \in \mathbb{Q}^+ \mid x \geq r \in \Gamma\}$ and $B = \{r \in \mathbb{Q}^+ \mid x \leq r \in \Gamma\}$ are both non-empty: Axiom (A2) guarantees that for any $r \in \mathbb{Q}^+$, either $x \geq r \in \Gamma$ or $x \leq r \in \Gamma$. Suppose that there exists $r \in \mathbb{Q}^+$ such that $x \geq r \in \Gamma$. Then, $A \neq \emptyset$. Suppose that $B = \emptyset$, then (A2) implies that for any $r \in \mathbb{Q}^+$, $x \geq r \in \Gamma$. Using (R3) for $C[X] = X$, we derive that $\bot \in \Gamma$ - contradiction. Consequently, $B \neq \emptyset$. Similarly can be proven that $B \neq \emptyset$ implies $A \neq \emptyset$.

Since the two sets are non-empty, the sup and inf exist. Suppose that $\sup A = \infty$. Then applying (A3) we obtain that $\mathbb{Q}^+ \subseteq A$ and (R2) for $C[X] = X$ proves the inconsistency of $\Gamma$ - contradiction. Similarly one can prove that $\inf B \in \mathbb{R}^+$.

Let $\sup A = u$ and $\inf B = v$. We prove $u = v$. If $u < v$, there exists $r \in \mathbb{Q}^+$ such that $u < r < v$. Since $u < r$, $x \geq r \notin \Gamma$ and (A2) guarantees that $x \leq r \in \Gamma$. But this contradicts the fact that $r \leq v$. If $v < u$, there exists $r_1, r_2 \in \mathbb{Q}^+$ such that $v < r_1 < r_2 < u$. Since $r_1 > v$, $x \leq r_1 \in \Gamma$ for $i = 1, 2$ (applying (A3)), and similarly, $r_1 < u$ implies $x \geq r_1 \in \Gamma$ for $i = 1, 2$. Since $r_2 - r_1 > 0$, we apply (A3) and obtain $\bot x \geq r_2 \rightarrow \neg(x \leq r_1)$. This shows that $\Gamma$ is inconsistent - contradiction. Consequently, $u = v$.

**Proof.** [Proof of Lemma 6.5]

We say that a set $S \subseteq \Omega$ is coherent if there exists a set $T \subseteq \Im(K)$ and a bijection $\gamma : T \rightarrow S$ that satisfies the two conditions of Definition 6.4. $S$ is maximally-coherent if $T = \Im(K)$. Observe that the bijection that defines a maximally-coherent set is a coherent function in the sense of Definition 6.4.

We prove, using a transfinite induction, that any coherent set has a maximally-coherent extension. And this proves the existence of $\gamma$, because the singleton $\{\Lambda\}$ is coherent.

**I.** Firstly, observe that the set $S \subseteq \Omega$ is coherent iff for arbitrary $\Lambda_1, \Lambda_2 \in \Omega$, with $i_s = \mathcal{I}(\Lambda_s), s = 1, 2$,

$$\Lambda_1 + (i_2 - i_1) \subseteq \Lambda_2 \text{ and } \Lambda_2 + (i_1 - i_2) \subseteq \Lambda_1,$$

where for arbitrary $\Lambda' \in \Omega$ and $\delta : K \rightarrow \mathbb{R}$,

$$\Lambda' + \delta = \{ \phi + f_- / f_+ \mid f_-, f_+ : K \rightarrow \mathbb{R}, f_- \leq \delta \leq f_+ \}.$$ 

Moreover $\Lambda_1 + (i_2 - i_1) \subseteq \Lambda_2$ iff $\Lambda_2 + (i_1 - i_2) \subseteq \Lambda_1$. We prove it as follows:

Suppose $\Lambda_1 + (i_2 - i_1) \subseteq \Lambda_2$ but $\Lambda_2 + (i_1 - i_2) \not\subseteq \Lambda_1$, i.e., there exist $\psi \in \Lambda_2$, bijection $\sigma$ and $f_-, f_+ : \mathcal{I}(K) \rightarrow \mathbb{Q}$ s.t. $\psi + \sigma f_- / f_+ \notin \Lambda_1$. Since $\Lambda_1$ is
 maximal, \((\psi +_\sigma f_- / f_+) \in \Lambda_1\). So \(\neg \psi +_\sigma f_+ / f_- \in \Lambda_1\). Since \(-f_- \leq i_2 - i_1 \leq -f_+\) and \(\sigma\) is bijection, \((\neg \psi +_\sigma f_+ / f_-) +_{\sigma^{-1}} -f_+ / -f_- \in \Lambda_2\). Then \(\neg \psi \in \Lambda_2\) - contradiction!

Hence \(\Lambda_2 + (i_1 - i_2) \subseteq \Lambda_1\).

Similarly for the other direction.

II. If \(i = \mathcal{I}(\Lambda)\) and \(i' \in \mathfrak{S}(\mathcal{K})\), then there exists \(\Lambda' \in \Omega\) s.t. \(\mathcal{I}(\Lambda') = i'\) and \(\{\Lambda, \Lambda'\}\) is coherent, i.e., \(\Lambda + (i' - i) \subseteq \Lambda'\). To prove this, we firstly prove the following two properties which will also be used in IV:

(a) For arbitrary \(\phi \in \Lambda\), \(\sigma\) a bijection on \(\mathcal{K}\), and \(f_-, f_+ : \mathcal{K} \to \mathbb{Q}\) s.t. for any \(x \in \mathcal{K}(\phi)\), either \(f_-(x) = f_+(x) = 0\) or \(f_-(x) \leq (i' - i)(x) \leq f_+(x)\). Then,

\[
\vdash (\phi +_\sigma f_- / f_+) +_{\sigma^{-1}} -(f_- o_{\sigma^{-1}})/(f_+ o_{\sigma^{-1}}) \rightarrow \phi.
\]

(b) For any \(x \leq r \in L\),

\[
\{ (x \leq r) + f_- / f_+ \mid f_-, f_+ : \mathcal{K} \to \mathbb{Q}, f_- < 0 < f_+ \}\vdash x \leq r.
\]

**Proof of (a):** Induction on \(\phi\).

**The Case \(x \geq r\):**

\[
(x \geq r) +_\sigma f_- / f_+ = \begin{cases} 
\sigma(x) \geq (r + f_-(x)), r + f_-(x) \geq 0 \\
\sigma(x) \geq 0, \text{ otherwise}
\end{cases}
\]

(i). If \(r + f_-(x) \geq 0:\)

\[
((x \geq r) +_\sigma f_- / f_+) +_{\sigma^{-1}} -(f_- o_{\sigma^{-1}})/(f_+ o_{\sigma^{-1}}) = (\sigma(x) \geq (r + f_-(x))) +_{\sigma^{-1}} -(f_- o_{\sigma^{-1}})/(f_+ o_{\sigma^{-1}}) = x \geq r;
\]

(ii). If \(r + f_-(x) < 0:\)

\[
((x \geq r) +_\sigma f_- / f_+) +_{\sigma^{-1}} -(f_- o_{\sigma^{-1}})/(f_+ o_{\sigma^{-1}}) = (\sigma(x) \geq 0) +_{\sigma^{-1}} -(f_- o_{\sigma^{-1}})/(f_+ o_{\sigma^{-1}}) = x \geq f_-(x).
\]

\(r + f_-(x) < 0\) implies \(-f_-(x) > r\) implies \(x \geq r\).

So \(\vdash ((x \geq r) +_\sigma f_- / f_+) +_{\sigma^{-1}} -(f_- o_{\sigma^{-1}})/(f_+ o_{\sigma^{-1}}) \rightarrow (x \geq r)\).

**The Case \(x \leq r\):** If \(f_-(x) = f_+(x) = 0\), then obviously

\[
((x \leq r) +_\sigma f_- / f_+) +_{\sigma^{-1}} -(f_- o_{\sigma^{-1}})/(f_+ o_{\sigma^{-1}}) = x \leq r.
\]

Otherwise,

\[
(x \leq r) +_\sigma f_- / f_+ = \begin{cases} 
\sigma(x) \leq (r + f_+(x)), r + f_+(x) \geq 0 \\
\sigma(x) \geq 0, \text{ otherwise}
\end{cases}
\]

\(x \leq r \in \Lambda, r \geq i_\Lambda(x)\), so \(r + f_+(x) \geq i_\Lambda(x) + \delta(x) = i(x) \geq 0\).

Then \((x \leq r) +_\sigma f_- / f_+ = \sigma(x) \leq (r + f_+(x))\), which implies \((x \leq r) +_\sigma f_- / f_+) +_{\sigma^{-1}} -(f_- o_{\sigma^{-1}})/(f_+ o_{\sigma^{-1}}) = x \leq r\).

So \(\vdash ((x \leq r) +_\sigma f_- / f_+) +_{\sigma^{-1}} -(f_- o_{\sigma^{-1}})/(f_+ o_{\sigma^{-1}}) \rightarrow (x \leq r)\).

**The Case \(x > r \text{ and } x < r\):** similar as the above two.

**The Case \(\phi \land \psi\) and \(\phi \lor \psi\):** Obviously.

**The Case \([a] \phi\):** \([(a) \phi] +_\sigma f_- / f_+) +_{\sigma^{-1}} -(f_- o_{\sigma^{-1}})/(f_+ o_{\sigma^{-1}}) \Rightarrow [a]((\phi +_\sigma f_- / f_+) +_{\sigma^{-1}} -(f_- o_{\sigma^{-1}})/(f_+ o_{\sigma^{-1}})). \quad \text{By inductive hypothesis;}

\[
\vdash (\phi +_\sigma f_- / f_+) +_{\sigma^{-1}} -(f_- o_{\sigma^{-1}})/(f_+ o_{\sigma^{-1}}) \rightarrow \phi. \quad \text{By (R1) and (A5)}
\]

\[
\vdash [a]((\phi +_\sigma f_- / f_+) +_{\sigma^{-1}} -(f_- o_{\sigma^{-1}})/(f_+ o_{\sigma^{-1}})) \rightarrow [a] \phi.
\]

Hence \(\vdash (([a] \phi) +_\sigma f_- / f_+) +_{\sigma^{-1}} -(f_- o_{\sigma^{-1}})/(f_+ o_{\sigma^{-1}}) \Rightarrow [a] \phi\).

**The Case \((a) \phi\):** \(((a) \phi) +_\sigma f_- / f_+) +_{\sigma^{-1}} -(f_- o_{\sigma^{-1}})/(f_+ o_{\sigma^{-1}}) = \langle a \rangle([\phi +_\sigma f_- / f_+] +_{\sigma^{-1}} -(f_- o_{\sigma^{-1}})/(f_+ o_{\sigma^{-1}})). \quad \text{By (R1) and (A5)}

\[
\vdash [a]((\phi +_\sigma f_- / f_+) +_{\sigma^{-1}} -(f_- o_{\sigma^{-1}})/(f_+ o_{\sigma^{-1}})) \rightarrow [a] \phi.
\]

Hence \(\vdash ([a] \phi) +_\sigma f_- / f_+) +_{\sigma^{-1}} -(f_- o_{\sigma^{-1}})/(f_+ o_{\sigma^{-1}}) \Rightarrow [a] \phi\).
∃ The Case \[\vdash \neg \exists x_{\in K} \] Then \(\neg x_{\in K}\) +_{\sigma-1} (f_{\circ \sigma-1})/(f_{\circ \sigma-1}) \rightarrow \phi. So \(\neg \phi \rightarrow \neg((\phi +_{\sigma} f_{\circ \sigma-1}) +_{\sigma-1} (f_{\circ \sigma-1})/(f_{\circ \sigma-1})).\)

By (R1) and (A5), \(\exists [a] \neg \phi \rightarrow [a] \neg((\phi +_{\sigma} f_{\circ \sigma-1}) +_{\sigma-1} (f_{\circ \sigma-1})/(f_{\circ \sigma-1})).\)

Then \(\exists [a] \neg((\phi +_{\sigma} f_{\circ \sigma-1}) +_{\sigma-1} (f_{\circ \sigma-1})/(f_{\circ \sigma-1}) \rightarrow \exists [a] \phi).\)

Hence \((\exists [a] \phi) +_{\sigma} f_{\circ \sigma-1} +_{\sigma-1} (f_{\circ \sigma-1})/(f_{\circ \sigma-1}) \rightarrow \exists [a] \phi).\)

[The Case \(\forall \phi)\]: \((\forall \phi) +_{\sigma} f_{\circ \sigma-1} +_{\sigma-1} (f_{\circ \sigma-1})/(f_{\circ \sigma-1}) = \forall x_{\phi} \rightarrow \exists x \phi \rightarrow \exists x \phi \rightarrow \forall x \phi.\)

Hence \((\forall \phi) +_{\sigma} f_{\circ \sigma-1} +_{\sigma-1} (f_{\circ \sigma-1})/(f_{\circ \sigma-1}) \rightarrow \forall x \phi.\)

[The Case \(\exists \phi)\]: \((\exists \phi) +_{\sigma} f_{\circ \sigma-1} +_{\sigma-1} (f_{\circ \sigma-1})/(f_{\circ \sigma-1}) = \exists x \phi \rightarrow \exists x \phi \rightarrow \exists x \phi.\)

Hence \((\exists \phi) +_{\sigma} f_{\circ \sigma-1} +_{\sigma-1} (f_{\circ \sigma-1})/(f_{\circ \sigma-1}) \rightarrow \exists x \phi.\)

[Proof of (b)]: [The Case \(x \geq r)\):

\[(x \geq r) +_{\sigma} f_{\circ \sigma-1} = \begin{cases} x \geq r +_{\sigma} f_{\circ \sigma-1} \geq 0, & \text{if } x \geq r, \\ x \geq 0, & \text{otherwise} \end{cases} \]

For any \(r +_{\sigma} f_{\circ \sigma-1} \geq 0, \text{ we have } x \geq (r + f_{\circ \sigma-1}). \) Since \(f_{\circ \sigma-1} < 0, \text{ we have exists } s \in \mathbb{Q} \) s.t. \(s = r + f_{\circ \sigma-1} < r \) and \(x \geq s. \) By rule (R2), \(x \geq r. \)

So \(\{x \geq r +_{\sigma} f_{\circ \sigma-1} \} \) for any \(f_{\circ \sigma-1} : \mathcal{K} \rightarrow \mathbb{Q} \) s.t. for any \(x \in \mathcal{K}, f_{\circ \sigma-1} < 0 < f_{\circ \sigma-1} \} \rightarrow (x \geq r). \)

[The Case \(x \leq r)]:
For any \( r + f_+ (x) \geq 0 \), we have \( x \leq (r + f_+ (x)) \). Since \( f_+ (x) > 0 \), we have exists \( s \in \mathbb{Q} \) s.t. \( s = r + f_+ (x) > r \), \( x \leq s \). By rule (R2), \( x \leq r \).

So \( \{ (x \leq r) + f_- / f_+ | \) for any \( f_- , f_+ : \mathcal{K} \to \mathbb{Q} \) s.t. for any \( x \in \mathcal{K} , f_- (x) < 0 < f_+ (x) \} \) \( \vdash (x \leq r) \).

**The Case** \( x > r \) and \( x < r \): similar as the above two.

Now we are ready to prove that there exists \( \Lambda' \in \Omega \) s.t. \( \mathcal{I} (\Lambda') = i' \) and \( \Lambda + (i' - i) \subseteq \Lambda' \).

Suppose not, i.e., for any \( \Lambda' \in \Omega \), either \( A \not\subseteq \Lambda' \) or \( A \subseteq \Lambda' \) but \( \mathcal{I} (\Lambda') \neq i' \), where \( A = \{ \phi +_\sigma f_+ | \phi \in \Lambda , f_- , f_+ : \mathcal{K} \to \mathbb{Q} \) s.t. \( f_- \leq i' - i \leq f_+ \} \).

* Suppose \( A \not\subseteq \Lambda' \), i.e., \( A \vdash \bot \) (otherwise by Rasiowa-Sikorski Lemma, there exists one maximal consistent set that contains it). Then \( B \vdash \bot \), where \( B = A +_{\sigma - 1} f_- o_{\sigma - 1} f_+ o_{\sigma - 1} = \{ \phi +_{\sigma - 1} f_- o_{\sigma - 1} f_+ o_{\sigma - 1} | \phi \in A \} \). For any \( \phi \in \Lambda , \phi +_{\sigma - 1} f_- o_{\sigma - 1} f_+ o_{\sigma - 1} \in B \). By (a), \( \phi \in B \). So \( \Lambda \subseteq B \). Since \( \Lambda \) is maximal, \( \Lambda = B \). So \( \Lambda \vdash \bot \) - contradiction!

* Suppose \( A \subseteq \Lambda' \) but \( \mathcal{I} (\Lambda') \neq i \).

First we prove \( x \leq r \not\in A \) implies \( - (x \leq r) \in A \) for any \( x \in \mathcal{K} \) and \( r \in \mathbb{Q}^+ \) as follows: \( x \leq r \not\in A \) implies \( (x \leq r) +_{\sigma - 1} - f_- o_{\sigma - 1} f_+ o_{\sigma - 1} \not\in A \). So \( -(x \leq r) +_{\sigma - 1} - f_- o_{\sigma - 1} f_+ o_{\sigma - 1} \in A \). Then \( -(x \leq r) +_{\sigma - 1} - f_- o_{\sigma - 1} f_+ o_{\sigma - 1} \in A \), where \( f_- , f_+ \leq i' - i \leq f_+ , f_- \) which implies \( f_- - f_+ \leq 0 \leq f_+ - f_- \). If \( i' - i \in \mathbb{Q}^+ \), then there exists \( f_- , f_+ \) s.t. \( f_- - f_+ = 0 \) (and \( f_- , f_+ \) s.t. \( f_+ - f_- = 0 \)). Hence \( -(x \leq r) \in A \). If not, \( f_- - f_+ < 0 < f_+ - f_- \).

By (b), \( -(x \leq r) \in A \).

Then for all \( r \geq i(x) \geq 0 \), suppose \( x > r \in \Lambda' \). So \( x > r \in A_i \) by the above result. Then \( x > r \not\in A \) which implies \( r - f_- (x) \leq i_A (x) \). Hence \( r \leq i_A (x) + f_- (x) \leq i(x) \) - contradiction! So \( x \leq r \in \Lambda' \).

For all \( 0 < r \leq i(x) \), suppose \( x < r \in \Lambda' \). Then \( x < r \in A_i \) by the above result. Then \( x < r \not\in A \), which implies \( r - f_+ (x) \geq i_A (x) \). Then \( r \geq i_A (x) + f_+ (x) \geq i(x) \) - contradiction! So \( x \geq r \in \Lambda' \).

If \( r = 0 \), apparently \( x \geq 0 \in \Lambda' \).

So \( \mathcal{I} (\Lambda') = i - contradiction! \)

**IV.** If \( S = \{ A_s | s \in \mathbb{N} \} \) is a countable coherent set (supp. \( \Lambda = \Lambda_0 \)) and \( j \in \exists (\mathcal{K}) \). There exists \( \Lambda_j \in \Omega \) s.t. \( S \cup \{ \Lambda_j \} \) is coherent. As before, we need to prove that

\[
X = \bigcup_{s \in \mathbb{N}} (A_s + (j - i_s))
\]

is consistent. Let \( X_t = \bigcup_{s \leq t} (A_s + (j - i_s)) \) for \( t \in \mathbb{N} \). We have previously proved that \( X_t \) is consistent for any \( t \in \mathbb{N} \). Moreover, observe that \( X = \bigcup_{t \in \mathbb{N}} X_t \).

Suppose that \( X \) is not consistent. If the contradiction can be proven without involving the infinitary rules, then there must exist one \( X_t \) that contains all the formulas in the proof, which contradicts the consistency of \( X_t \). Consequently, if \( X \) is inconsistent it must be because, when \( t \) goes to infinite, the sets \( X_t \setminus X_{t-1} \) collect the
left-hand side of some of our infinitary rules.

For arbitrary contexts $C$, formulas $\phi \in L$, $r \in \mathbb{Q}_{\geq 0}$ and $x \in K$ we denote by $L_1(C, x) = \{C[x \leq s], C[x \geq s] \mid s \in \mathbb{Q}_{\geq 0}\}$, $L_2(C, x, \phi) = \{C[\phi + [x \mapsto s]/[x \mapsto s']] \mid s, s' \in \mathbb{Q}, s \leq s'\}$, $L_3(C, x, \phi, r) = \{C[\forall(r \leq x \leq s \rightarrow \phi)] \mid r, s \in \mathbb{Q}_{\geq 0}, r \leq s\}$.

We already proved that for each $t \in \mathbb{N}$, $X_t$ is $L_1(C, x)$-maximally consistent in II. Similarly, for $L_2(C, x, \phi)$ either $X_t \cap L_2(C, x, \phi) = \emptyset$ or $X_t$ is $L_2(C, x, \phi)$-maximally consistent. To prove the same result for $L_3(C, x, \phi)$, we rely on the following fact:

If $\Lambda_1, \Lambda_2$ are coherent and $t_1 = \sup\{s \in \mathbb{Q} \mid \forall(x \leq s \rightarrow \phi) \in \Lambda_1\}$, then there exists no $t_2 > t_1 + (i_2(x) - i_1(x))$ such that for any $s' \leq t_2$ and any $f_- \leq f_1 - i_1 \leq f_+$, $\forall(x \leq s' \rightarrow (\phi + f_- / f_+)) \in \Lambda_2$.

This demonstrates that if in $X$ we can use some infinitary rule, it must be used within some $X_t$ for some $t \in \mathbb{N}$. Consequently, the consistency of $X_t$ proves the consistency of $X$.

V. If $S \subseteq \Omega$ is an uncountable coherent set and $j \in \exists(K)$, then there exists $\Lambda_j \in \Omega$ s.t. $S \cup \{j\}$ is coherent. Supp. $S = \{\Lambda_s \mid s \in \mathbb{R}\}$, $\Lambda = \Lambda_0$ and $i_s = \mathcal{J}(\Lambda_s)$. As in the other cases it is sufficient to prove that the set $Y = \bigcup_{s \in \mathbb{R}}(\Lambda_s + (j - i_s))$ is consistent. Since $Y \subseteq L$ and $L$ is countable, $Y$ is countable and consequently we can assume that $Y = \bigcup_{s \in \mathbb{N}}(\Lambda_s + (j - i_s))$.

Further the result is a consequence of IV. \hfill \Box

Proof. [Proof of Theorem 6.6] We need to prove that $\oplus$ is well-defined and satisfies the required conditions 1 and 2 in Definition 3.1. That is to prove the following three conditions:

1. $\oplus$ is a well-defined partial function: $\mathbb{R}_{\geq 0} \times \Gamma \rightarrow \Gamma$, i.e. if $\gamma_1 = d \oplus \gamma$ and $\gamma_2 = d \oplus \gamma$, then $\gamma_1 = \gamma_2$.

For any $i \in \mathcal{J}(K)$, $\phi \in \gamma_1(i)$ implies $x = [d] \wedge \phi \in \gamma_1(i[x \mapsto [d]])$, where $x \in K \setminus K(\phi)$ is a new clock different from those in $\phi$. It implies that $\forall(x = [d] \rightarrow \phi) \in \gamma(i[x \mapsto [d]]) - d) \because \gamma_1 = d \oplus \gamma$ and axiom (A10). So $x = [d] \rightarrow \phi \in \gamma_2(i[x \mapsto [d]])$ because $\gamma_2 = d \oplus \gamma$. Then $\phi \in \gamma_2(i[x \mapsto [d]])$, which implies $\phi \in \gamma_2(i)$ since $x \in K \setminus K(\phi)$.

Hence for any $i \in \mathcal{J}(K)$, $\gamma_1(i) \subseteq \gamma_2(i)$. Similarly $\gamma_2 \subseteq \gamma_2(i)$. Then for any $i \in \mathcal{J}(K)$, $\gamma_1(i) = \gamma_2(i)$. Hence $\gamma_1 = \gamma_2$.

2. For any $\gamma \in \Gamma$, $0 \oplus \gamma = \gamma$, i.e. for any $i \in \mathcal{J}(K)$, $\forall \phi \in \gamma$ implies $\phi \in \gamma$. It is obviously true by (A9).

3. $d \oplus (d' \oplus m) \equiv (d + d') \oplus m$, i.e. $\exists \gamma_1, \gamma_2$ s.t. $\gamma_1 = d' \oplus \gamma$, $\gamma_2 = d \oplus \gamma$ if $\exists \gamma_3$ s.t. $\gamma_3 = (d + d') \oplus \gamma$. 210
(⇒): suppose ∃γ₁, γ₂ s.t. γ₁ = d ⊕ γ, γ₂ = d' ⊕ γ₁. Then for any i ∈ 𝔐(𝔉), for any
∀φ ∈ γ(i) implies ∀∀φ ∈ γ(i) by (A8). So ∀φ ∈ γ₁(i + d) because γ₁ = d ⊕ γ,
which further implies φ ∈ γ₂(i + d + d') because γ₂ = d' ⊕ γ₁. Hence ∃γ₃ = γ₂ s.t.
γ₃ = (d + d') ⊕ γ.

(⇐): suppose ∃γ₃ s.t. γ₃ = (d + d') ⊕ γ.

- First, we prove: if γ₃ = (d + d') ⊕ γ then ∃γ₁ s.t. γ₁ = d' ⊕ γ.

For any i ∈ 𝔐(𝔉), let λᵢ = {φ | ∀∀φ ∈ γ(i)}, Δᵢ = ⋃ₓ∈𝔉{x ≤ r | i(x) ≤ r, r ∈ ℚ⁺},
Θᵢ₊d' = {φ | λᵢ ∪ Δᵢ₊d' ⊢ φ}.

We construct γ₁ according to the following steps:

(1) We prove Θᵢ₊d' is consistent.

We only need to show that λᵢ ∪ Δᵢ₊d' is consistent.

Suppose not, i.e. λᵢ ∪ Δᵢ₊d' ⊢ ⊥.

Since both λᵢ and Δᵢ₊d' are consistent, exists ρ ∈ 𝔉 s.t. λᵢ ⊢ ¬ρ and Δᵢ₊d' ⊢ ρ.
λᵢ ⊢ ¬ρ implies ∀λᵢ ⊢ ∀¬ρ by rule (R2) or (R4), which further implies ∀¬ρ ∈ γ(i).
Δᵢ₊d' ⊢ ρ implies ρ ∈ γ'(i + d') for any γ' ∈ Γ.

Let δ = (i + d')[K(ρ) → [d']'] - (i + d'), f₋, f₊ : 𝔉 → ℚ defined as:

\[ f₋(x) = \begin{cases} [d'] - rₓ, & x ∈ K(ρ) \\ 0, & \text{otherwise} \end{cases} \]

\[ f₊(x) = \begin{cases} [d'] - sₓ, & x ∈ K(ρ) \\ 0, & \text{otherwise} \end{cases} \]

where \( rₓ = \max\{x ≥ rₓ \text{ is in } ρ \text{ with } ¬ \text{ on atoms}\}, \) \( sₓ = \min\{x ≤ sₓ \text{ is in } ρ \text{ with } ¬ \text{ on atoms}\}. \)

Then ρ + f₋/f₊ ∈ γ'((i + d')[K(ρ) → [d']']) which implies ρ + f₋/f₊ ∈ γ'(i'[K(ρ) → [d']']) for any i' ∈ 𝔐(𝔉). Then \( \wedge_{x∈K(ρ)}(x = [d']) \rightarrow ρ + f₋/f₊ ∈ γ'(i') \) for any γ' and i'. By rule (R2) \( \forall(\wedge_{x∈K(ρ)}(x = [d']') \rightarrow ρ + f₋/f₊) ∈ γ'(i'). \)

\[ x ≤ [d'] ∈ γ(i[K(ρ) → [d']' - d']) \] for any \( x ∈ K(ρ). \) Then \( \vee_{x∈K(ρ)}(x ≤ [d']') \) ∈ \( γ(i[K(ρ) → [d']' - d']) \) and similarly \( \vee_{x∈K(ρ)}(x ≥ [d']') \) ∈ \( γ(i[K(ρ) → [d']' - d']) \). By axiom (A11) \( \exists(\vee_{x∈K(ρ)}(x = [d']')) ∈ γ(i[K(ρ) → [d']' - d']). \) So \( \vee_{x∈K(ρ)}(x ≤ [d']') \wedge \exists(\vee_{x∈K(ρ)}(x ≥ [d']')) \in γ(i[K(ρ) → [d']' - d']). \)

Together with \( \forall(\wedge_{x∈K(ρ)}(x = [d']') \rightarrow ρ + f₋/f₊) ∈ γ(i[K(ρ) → [d']']) \), we get \( \exists(ρ + f₋/f₊) ∈ γ(i[K(ρ) → [d']']) \) by axiom (A10).

So \( ∃ρ ∈ γ(i) - \text{ contradict that } ∀¬ρ ∈ γ(i). \) Hence λᵢ ∪ Δᵢ₊d' is consistent, which implies Θᵢ₊d' is consistent.

211
(2) Extend to maximal consistent set $\Lambda_i$ for each $i \in I$.

(i) If there exists $x \in \mathcal{K}$ s.t. $i(x) \geq d'$ and $i(x) \in \mathbb{Q}^+$, $\Theta_i$ is maximal, i.e. for any $\phi \in \mathcal{L}$, $\phi \not\in \Theta_i$ implies $\neg\phi \in \Theta_i$.

By definition of $\Delta_i$, $x = i(x) \in \Delta_i \subseteq \Theta_i$. Together with $\phi \not\in \Theta_i$, $x = i(x) \rightarrow \phi \not\in \Theta_i$.

By definition of $\lambda_{i-d'}$, $\forall (x = i(x) \rightarrow \phi) \in \gamma(i-d')$ implies $x = i(x) \rightarrow \phi \in \lambda_{i-d'} \subseteq \Theta_i$. So $\forall (x = i(x) \rightarrow \phi) \not\in \gamma(i-d')$. Since $\gamma(i-d')$ is maximal and $\neg(\exists (x = i(x) \wedge \neg\phi)) = \forall (x = i(x) \rightarrow \phi)$, we have $\forall (x = i(x) \rightarrow \phi) \not\in \gamma(i-d')$ implies $\exists (x = i(x) \wedge \neg\phi) \in \gamma(i-d')$. By (A10) $\forall (x = i(x) \rightarrow \neg\phi) \in \gamma(i-d')$. So $x = i(x) \rightarrow \neg\phi \in \lambda_{i-d'} \subseteq \Theta_i$ by the definition of $\lambda_{i-d'}$. Then we have $\neg\phi \in \Theta_i$.

So $\Lambda_i = \Theta_i$, if there exists $x \in \mathcal{K}$ s.t. $i(x) \geq d'$ and $i(x) \in \mathbb{Q}^+$.

(ii) If for all $x \in \mathcal{K}$, $i(x) \geq d'$ and $i(x) \in \mathbb{R}_{\geq 0}$ but $\not\in \mathbb{Q}^+$, define the following function, which gives an interpretation and finite set of clocks $\mathcal{K}_f$, maps to another interpretation:

$$i_{\mathcal{K}_f} = \begin{cases} i(x) & x \in \mathcal{K}_f \\ [d'] & \text{otherwise} \end{cases}$$

Then we define $\Lambda_i = \bigcup_{\mathcal{K}_f \in \mathcal{K}} \{ \phi \mid \mathcal{K}(\phi) = \mathcal{K}_f, \phi \in \Pi_{\mathcal{K}_f} \}$.

Now we prove that $\Lambda_i$ above is maximal consistent.

Obviously for any $x \in \mathcal{K}_f$ and any finite $\mathcal{K}_f \subseteq \mathcal{K}$, $i_{\mathcal{K}_f}(x) - d' \geq 0$. And according to the first case, $\Lambda_{\mathcal{K}_f}$ is maximal consistent. So $\Lambda_i$ is maximal.

Suppose $\Lambda_i$ is not consistent, i.e. $\Lambda_i \vdash \bot$. Suppose $\Psi \subseteq \Lambda_i$ s.t. $\Psi \vdash \bot$, and $\mathcal{K}(\Psi)$ is the set of clocks of $\Psi$. Since even in the infinitary rules, the number of the clocks is finite, $\mathcal{K}(\Psi)$ is finite. For any $\psi \in \Psi$, $\psi \in \Lambda_{\mathcal{K}(\psi)}$ by the definition of $\Lambda_i$.

By the first case, $\Lambda_{\mathcal{K}(\psi)} = \Theta_{\mathcal{K}(\psi)}$, so $\forall (\psi' \in \mathcal{K}(\psi)) \vdash \psi$. Suppose $\Psi' \subseteq \lambda_{\mathcal{K}(\psi)} - d' \cup \Delta_{\mathcal{K}(\psi)}$ s.t. $\Psi' \vdash \bot$. $\forall (y \in \mathcal{K}(\psi)) \vdash \psi'$ for any $y \in \mathcal{K}(\Psi) \setminus \mathcal{K}(\psi), y \not\in \mathcal{K}(\Psi)$. Since $\Psi' \setminus \mathcal{K}(\psi)$ has no clocks in $\mathcal{K}(\Psi) \setminus \mathcal{K}(\psi)$, and $i_{\mathcal{K}(\psi)}$ and $i_{\mathcal{K}(\psi)}$ only differ for the clocks in $\mathcal{K}(\Psi) \setminus \mathcal{K}(\psi)$, it is easy to have $\Psi' \setminus \mathcal{K}(\psi) \subseteq \lambda_{\mathcal{K}(\psi)} - d' \cup \Delta_{\mathcal{K}(\psi)}$. Then $\psi \in \Theta_{\mathcal{K}(\psi)}$.

Hence $\Psi \subseteq \Theta_{\mathcal{K}(\psi)}$, which implies $\Theta_{\mathcal{K}(\psi)} \vdash \bot$ - contradiction.

(iii) If for all $x \in \mathcal{K}$, $i(x) < d'$.

$\Lambda_i = \{ \phi \mid \phi + [d'] \in \Lambda_{i+[d']} \}$.

Obviously $\Lambda_i$ is a consistent set since $\Lambda_{i+[d']}$ is consistent. Now we prove $\Lambda_i$ is also maximal. Suppose $\phi \not\in \Lambda_i$, so $\phi + [d'] \not\in \Lambda_{i+[d']}$, which implies $\neg(\phi + [d']) \in \Lambda_{i+[d']}$. Since $\Lambda_{i+[d']}$ is maximal, then we have $\neg\phi \in \Lambda_i$. So $\Lambda_i$ is maximal.

(3) Let $\gamma_1(i) = \Lambda_i$.

We prove that $\gamma_1$ is a coherent function, i.e., $\gamma_1$ satisfies the two conditions in Definition 6.4.

The first condition holds obviously.
Before we prove $\gamma_1$ satisfies the second condition, we first prove the following:

(a) For any bijection $\sigma$ on $\mathcal{K}$, any $\delta : \mathcal{K} \to \mathbb{R}$ s.t. $\forall x \in \mathcal{K}, [i(x) \geq d', i(x) + \delta(x) \geq d']$, and any $f_-, f_+ : \mathcal{K} \to \mathbb{Q}$ s.t. $f_- \leq \delta \leq f_+$, we have: $\phi \in \Theta_i \Rightarrow \phi + \sigma f-/f_+ \in \Theta_{(i+\delta)\sigma^{-1}}$.

$\phi \in \Theta_i$ implies $\lambda_{i-d'} \cup \Delta_i \vdash \phi$. Because $\gamma$ is interpretation coherent, so $\lambda_{i-d'} + \sigma f-/f_+ \subseteq \lambda_{(i+\delta)\sigma^{-1}-d'} \cdot \Delta_{i} + \sigma f-/f_+ \subseteq \Delta_{(i+\delta)\sigma^{-1}}$ by the definition of $\Delta_i$. So we have $\lambda_{(i+\delta)\sigma^{-1}-d'} \cup \Delta_{(i+\delta)\sigma^{-1}} \vdash \phi + \sigma f-/f_+$. Hence $\phi \in \Theta_i \Rightarrow \phi + \sigma f-/f_+ \in \Theta_{(i+\delta)\sigma^{-1}}$.

(b) For any bijection $\sigma$ on $\mathcal{K}$, any $\delta : \mathcal{K} \to \mathbb{R}$ s.t. $\forall x \in \mathcal{K}, [i(x) \geq d', i(x) + \delta(x) \geq d']$, and any $f_-, f_+ : \mathcal{K} \to \mathbb{Q}$ s.t. $f_- \leq \delta \leq f_+$, we have: $\phi \in \Lambda_i \Rightarrow \phi + \sigma f-/f_+ \in \Lambda_{(i+\delta)\sigma^{-1}}$.

If $\Lambda_i = \Theta_i$, it’s the (a) case.

If not, $\phi \in \Lambda_{i\mathcal{K}(\phi)} = \Theta_{i\mathcal{V}(\phi)}$. $i(x) + \delta(x) \geq 0$ implies $i^{\mathcal{K}(\phi)}(x) + \delta^{\mathcal{K}(\phi)}(x) \geq 0$, so by the first case, $\phi + \sigma f-/f_+ \in \Theta_{(i\mathcal{K}(\phi)+\delta^{\mathcal{K}(\phi)})\sigma^{-1}}$.

$(i^{\mathcal{K}(\phi)} + \delta^{\mathcal{K}(\phi)}) \circ \sigma^{-1}(x) = i^{\mathcal{K}(\phi)}(\sigma^{-1}(x)) + \delta^{\mathcal{K}(\phi)}(\sigma^{-1}(x)) = (i \circ \sigma^{-1})^{\mathcal{K}(\phi)}(\sigma^{-1}(x)) + (\delta \circ \sigma^{-1})^{\mathcal{K}(\phi)}(\sigma^{-1}(x))$ (because $\sigma^{-1}(x) \in \mathcal{K}(\phi)$ iff $x \in \mathcal{K}(\phi + \sigma f-/f_+)$).

So we have $\phi + \sigma f-/f_+ \in \Theta_{((i+\delta)\sigma^{-1})\mathcal{K}(\phi+f-/f_+)}$.

Hence $\phi \in \Lambda_i \Rightarrow \phi + \sigma f-/f_+ \in \Lambda_{(i+\delta)\sigma^{-1}}$.

Now we are ready to prove the second condition: for any bijection $\sigma$ on $\mathcal{K}$, any $\delta : \mathcal{K} \to \mathbb{R}$ s.t. $\forall x \in \mathcal{K}, [i(x) + \delta(x) \geq 0]$, and any $f_-, f_+ : \mathcal{K} \to \mathbb{Q}$ s.t. $f_- \leq \delta \leq f_+$, we have: $\phi \in \gamma(i) \Rightarrow \phi + \sigma f-/f_+ \in \gamma((i+\delta)\sigma^{-1})$.

- If $i \geq d'$ and $i + \delta \geq d'$, it’s the (b) case above.

- If $i \not\geq d'$ and $i + \delta \geq d'$:

  $\phi + [d'] \in \gamma_1(i + [d'])$ by the definition of $\gamma_1$. Let $i' = i + [d']$, $\delta' = \delta - [d']$, $f_- = f_- - [d']$ and $f_+ = f_+ - [d']$. Then we have $i' \geq d', i' + \delta' = i + \delta \geq d'$. So we have $(\phi + [d']) + \sigma f_-/f_+ \in \gamma_1((i' + \delta')\sigma^{-1})$, which implies $\phi + \sigma f-/f_+ \in \gamma((i+\delta)\sigma^{-1})$.

- If $i \geq d'$ and $i + \delta \not\geq d'$:

  Then $\phi + \sigma f_- + [d'/f_+] \in \gamma_1((i + \delta + [d'])\sigma^{-1})$.

  Induction on $\phi$:

  **[The case $x \geq r$]**:

  $$(x \geq r) + \sigma f-/f_+ + [d'] =
  \begin{cases}
  \sigma(x) \geq r + f_-(x) + [d'], & \text{if } r + f_-(x) \geq 0 \\
  \sigma(x) \geq [d'], & \text{otherwise}
  \end{cases}$$
So \((x \geq r) +_\sigma f^-/f_+ + [d'] \in \gamma_1((i + \delta + d') \circ \sigma^{-1}).\)

[The case \(x > r\): similar as above.

[The case \(x \leq r\):

\[
(x \leq r) +_\sigma f^-/f_+ + [d'] =
\begin{cases} 
\sigma(x) \leq r + f_+(x) + [d'], & \text{if } r + f_+(x) \geq 0 \\
\sigma(x) \leq [d'], & \text{otherwise}
\end{cases}
\]

So \((x \leq r) +_\sigma f^-/f_+ + [d'] \in \gamma_1((i + \delta + d') \circ \sigma^{-1}).\)

[The case \(x < r\): similar as above.

Other cases hold obviously.

So we have \(\phi +_\sigma f^-/f_+ \in \gamma_1((i + \delta) \circ \sigma^{-1}).\)

• If \(i \not\geq d'\) and \(i + \delta \not\geq d'\): 
  \(\phi \in \gamma_1(i)\) implies \(\phi + [d'] \in \gamma_1(i + [d']).\) Then \((\phi + [d']) +_\sigma f^-/f_+ \in \gamma_1((i + [d'] + \delta) \circ \sigma^{-1}).\) So \(\phi +_\sigma f^- + [d']/f_+ + [d'] \in \gamma_1((i + [d'] + \delta) \circ \sigma^{-1}).\) Then we can do the same deduction as the above case.

So we proved that there exists \(\gamma_1\) constructed as above, s.t. \(\gamma_1 = d' \oplus \gamma.\)

- Now we prove that: \(\exists \gamma_2 \text{ s.t. } \gamma_2 = d \oplus \gamma_1.\)

In the following we prove that \(\gamma_3 = d \oplus \gamma_1,\) i.e. for any \(i \in \mathcal{J}(\mathcal{K}), \forall \phi \in \gamma_1(i)\) implies \(\phi \in \gamma_3(i + d).\)

Secondly we need to prove that exists \(\gamma_2\) s.t. \(\gamma_2 = d \oplus \gamma_1.\) We prove that \(\gamma_3 = d \oplus \gamma_1,\) i.e. for any \(i \in \mathcal{J}(\mathcal{K}),\) for any \(\forall \phi \in \gamma_1(i)\) implies \(\phi \in \gamma_3(i + d).\)

For any \(i \in \mathcal{J}(\mathcal{K}), \forall \phi \in \gamma_1(i)\) implies \(x \leq r \wedge \forall \phi \in \gamma_1(i)\) for \(r \in \mathbb{Q}^+\) s.t. \(i(x) \leq r \leq i(x) + d.\) Then \(x \leq r + [d'] \wedge \forall \phi + [d'] \in \gamma_1(i + [d']).\) Since \(\gamma_1 = d' \oplus \gamma,\) we get \(x \leq r + [d'] \wedge \forall \phi + [d'] \in \gamma(i + [d'] - d').\) By (A9), \(\forall x \geq r + [d'] \rightarrow \forall \phi + [d'] \in \gamma(i + [d'] - d').\) So \(\forall x \geq r + [d'] \rightarrow \forall \phi + [d'] \in \gamma_3(i + [d'] + d)\) since \(\gamma_3 = (d + d') \oplus \gamma.\) And \(r \leq i(x) + d,\) which implies \(x \geq r + [d'] \in \gamma_3(i + [d'] + d).\) Then \(\forall \phi + [d'] \in \gamma_3(i + [d'] + d).\) Hence \(\forall \phi \in \gamma_3(i + d).\)

So there exists \(\gamma_2 = \gamma_3\) s.t. \(\gamma_2 = d \oplus \gamma_1.\)

Hence \(W = (\Gamma, \theta, \oplus)\) defined above is a timed labeled transition system. \(\square\)
A Metrized Duality Theorem for Markov Processes

Dexter Kozen  Radu Mardare  Prakash Panangaden
Cornell University, USA  Aalborg University, Denmark  McGill University, Canada

Abstract
We extend our previous duality theorem for Markov processes by equipping the processes with a pseudometric and the algebras with a notion of metric diameter. We are able to show that the isomorphisms of our previous duality theorem become isometries in this quantitative setting. This opens the way to developing theories of approximate reasoning for probabilistic systems.

Keywords: Markov processes, Aumann algebras, Stone duality, metrics, isometry, probabilistic reasoning, quantitative reasoning.

1 Introduction

Stone-type dualities are recognized as being ubiquitous, especially in computer science. For example Plotkin [22] and Smyth [23] (see also [21]) emphasized that the duality between state-transformer semantics and predicate-transformer semantics is an instance of Stone-type duality. A similar duality was observed for probabilistic transition systems [14]. Recently several authors, see for example [3], have emphasized the duality between logics and transition systems from a coalgebraic perspective. Mislove et al. [19] have found a duality between labelled Markov processes and $C^\ast$-algebras based on the closely related classical Gelfand duality.

In a recent paper [15], a Stone-type duality was developed for Markov processes defined on continuous state spaces. The algebraic counterpart of the Markov processes were called Aumann algebras in honour of Aumann’s work on probabilistic reasoning [1]. Aumann algebras capture, in algebraic form, a modal logic in which bounds on probabilities enter into the modalities. This logic can be stripped down to a very spartan core – just the modalities and finite conjunction – and still characterize bisimulation for labelled Markov processes [5, 6]. However, to obtain
the strong completeness properties that are implied by the duality theorems, one needs infinitary proof principles [12].

One of the critiques [11] of logics and equivalences used for the treatment of probabilistic systems is that boolean logic is not robust with respect to small perturbations of the real-valued system parameters. Accordingly, a theory of metrics [7, 8, 24, 25] was developed and metric reasoning principles were advocated. In the present paper we extend our exploration of duality theory with an investigation into the role of metrics and exhibit a metric analogue of the duality theory. This opens the way for an investigation into quantitative aspects of approximate reasoning.

In the present paper we integrate quantitative information into the duality of [15] by endowing Markov processes with a (pseudo)metric and Aumann algebras with a quantitative “norm-like” structure called a metric diameter. The interplay between the pseudometric and the boolean algebra is somewhat delicate and had to be carefully examined for the duality to emerge. The final results have easy proofs but the correct way to impose quantitative structure on Aumann algebras was elusive. The key idea is to axiomatize the notion of metric diameter on the Aumann algebra side. This is a concept more like a norm than a distance, but one can derive a metric from it. The idea comes from a paper by Banaschewski and Pultr [2] on Stone duality for metric spaces. However, our formulation is not quite the same as theirs.

2 Background

Given a relation $\mathcal{R} \subseteq M \times M$, the set $N \subseteq M$ is $\mathcal{R}$-closed iff

$$\{m \in M \mid \exists n \in N, (n, m) \in \mathcal{R}\} \subseteq N.$$ 

We assume that the reader is familiar with the definitions of field of sets, $\sigma$-algebra, measurable function, measurable space and measure.

If $(M, \Sigma)$ is a measurable space and $\mathcal{R} \subseteq M \times M$ is a relation on $M$, then $\Sigma(\mathcal{R})$ denotes the set of measurable $\mathcal{R}$-closed subsets of $M$.

Given a measurable space $\mathcal{M} = (M, \Sigma)$, we view the set $\Delta(M, \Sigma)$ of measures defined on $\Sigma$, as itself being a measurable space by endowing it with the $\sigma$-algebra $\mathfrak{F}$ generated by the sets $F(S, r) = \{\mu \in \Delta(M, \Sigma) : \mu(S) \geq r\}$ for arbitrary $S \in \Sigma$ and $r > 0$.

Measure theory works most smoothly in conjunction with certain topological assumptions. A Polish space is the topological space underlying a complete separable metric space. An analytic space is the image of a Polish space $X$ under a continuous function $f : X \to Y$, where $Y$ is also a Polish space. The special properties of analytic spaces were crucial in the proof of the logical characterization of bisimulation [6].

Let $M$ be a set and $d : M \times M \to \mathbb{R}$.
Definition 2.1 We say that \( d \) is a pseudometric on \( M \) if for arbitrary \( x, y \in M \),

\[
\begin{align*}
(1): & \quad d(x, x) = 0 \\
(2): & \quad d(x, y) = d(y, x) \\
(3): & \quad d(x, y) \leq d(x, z) + d(z, y)
\end{align*}
\]

We say that \((M, d)\) is a pseudometric space.

Pseudometrics arise as metric analogues of bisimulation \([7, 8]\). A pseudometric defines an equivalence relation called the kernel of the pseudometric, by \( x \sim y \) iff \( d(x, y) = 0 \). The metrics defined in \([7, 8]\) had bisimulation as the kernel.

In a pseudometric space \((M, d)\), the open ball with center \( x \in M \) and radius \( \varepsilon > 0 \) is the set \( \{ y \in M \mid d(x, y) < \varepsilon \} \). The collection of open balls forms a base for a topology called the metric topology. We can extend the pseudometric to sets in a manner analogous to the way in which one extends metrics to compact sets.

Definition 2.2 For a pseudometric \( d \) on \( M \) we define the Hausdorff pseudometric \( d^H \) on the class of subsets of \( X \) by

\[
d^H(X, Y) = \max(\sup_{x \in X} \inf_{y \in Y} d(x, y), \sup_{y \in Y} \inf_{x \in X} d(x, y)).
\]

We need to verify that these are indeed pseudometrics. The proof of the following lemma is omitted here; it is not too hard.

Lemma 2.3 If \( d : M \times M \to [0, 1] \) is a pseudometric on \( M \), then \( d^H \) is a pseudometric on subsets of \( M \).

Markov processes (MPs) are models of probabilistic systems with a continuous state space and probabilistic transitions \([6, 9, 20]\). In earlier papers, they were called labeled Markov processes to emphasize the fact that there were multiple possible actions, but here we will suppress the labels, as they do not contribute any relevant structure for our results.

Definition 2.4 [Markov process] A Markov process (MP) is a tuple \( \mathcal{M} = (M, \Sigma, \theta) \), where \((M, \Sigma)\) is an analytic space and \( \theta \in \mathbb{M} \to \Delta(M, \Sigma) \).

In a Markov process \( \mathcal{M} = (M, \Sigma, \theta) \), \( M \) is the support set, denoted by \( \text{supp}\mathcal{M} \), and \( \theta \) is the transition function. For \( m \in M \), \( \theta(m) : \Sigma \to [0, 1] \) is a probability measure on the state space \((M, \Sigma)\). For \( N \in \Sigma \), the value \( \theta(m)(N) \in [0, 1] \) represents the probability of a transition from \( m \) to a state in \( N \).

The condition that \( \theta \) is a measurable function \([M \to \Delta(M, \Sigma)]\) is equivalent to the condition that for fixed \( N \in \Sigma \), the function \( m \mapsto \theta(m)(N) \) is a measurable function \([M \to [0, 1]]\) (see e.g. Proposition 2.9 of \([9]\)).
Markovian logic (ML) is a multi-modal logic for semantics based on MPs [1, 4, 10, 13, 16, 18, 26]. In addition to the Boolean operators, this logic is equipped with probabilistic modal operators $L_r$ for $r \in \mathbb{Q}_0$ that bound the probabilities of transitions. Intuitively, the formula $L_r \varphi$ is satisfied by $m \in \mathcal{M}$ whenever the probability of a transition from $m$ to a state satisfying $\varphi$ is at least $r$.

**Definition 2.5** [Syntax] The formulas of $\mathcal{L}$ are defined by the grammar

$$\varphi ::= \bot | \varphi \to \varphi | L_r \varphi$$

where $r$ can be any element of $\mathbb{Q}_0$.

The Boolean operators $\lor$, $\land$, $\neg$, and $\top$ are defined from $\to$ and $\bot$ as usual. For $r_1, \ldots, r_n \in \mathbb{Q}_0$ and $\varphi \in \mathcal{L}$, let

$$L_{r_1 \cdots r_n} \varphi = L_{r_1} \cdots L_{r_n} \varphi.$$

The semantics for $L_r \varphi$ is defined as follows; the semantics of the other constructs are obvious. For MP $\mathcal{M} = (M, \Sigma, \theta)$ and $m \in M$,

$$\mathcal{M}, m \models L_r \varphi \text{ if } \theta(m)([\varphi]) \geq r,$$

where $[\varphi] = \{ m \in M \mid \mathcal{M}, m \models \varphi \}$.

For this to make sense, $[\varphi]$ must be measurable, this is readily verified. We define negation in the obvious fashion and use the words valid and satisfiable in the usual way.

This logic can be axiomatized using Hilbert-style axioms. The (strong) completeness of this logic is proved in [12, 18, 26] by assuming Lindenbaum’s lemma (every consistent set can be expanded to a maximal consistent set) as a meta-axiom. The duality theorem of Kozen et al. [15] implies strong completeness without needing this assumption.

The logical equivalence induced by ML on the class of MPs coincides with bisimulation equivalence [6, 20]. The proof requires that the state space be an analytic space.

### 3 Stone Duality for Markov Processes

In this section we briefly summarize the results of our previous duality paper [15].

We introduce an algebraic version of Markovian logic consisting of a Boolean algebra with operators $F_r$ for $r \in \mathbb{Q}_0$ corresponding to the operators $L_r$ of ML. We call these algebras Aumann algebras. They are dual to certain Markov processes constructed from zero-dimensional Hausdorff spaces called Stone–Markov processes (SMPs).
Definition 3.1 [Aumann algebra] An Aumann algebra (AA) is a structure \( \mathcal{A} = (A, \rightarrow, \bot, \{F_r\}_{r \in \mathbb{Q}_0}, \leq) \) where

- \((A, \rightarrow, \bot, \leq)\) is a Boolean algebra;
- for each \( r \in \mathbb{Q}_0 \), \( F_r : A \to A \) is a unary operator; and
- the axioms in Table 1 hold for all \( a, b \in A \) and \( r, s, r_1, \ldots, r_n \in \mathbb{Q}_0 \).

The Boolean operations \( \lor, \land, \neg, \top \) are defined from \( \rightarrow \) and \( \bot \) as usual.

Morphisms of Aumann algebras are Boolean algebra homomorphisms that commute with the operations \( F_r \). The category of Aumann algebras and Aumann algebra homomorphisms is denoted \( \text{AA} \). We abbreviate \( F_{r_1} \cdots F_{r_n} a \) by \( F_{r_1 \cdots r_n} a \).

### Aumann algebra

\begin{align*}
\text{(AA1)} \quad \top &\leq F_0 a \\
\text{(AA2)} \quad \top &\leq F_r \top \\
\text{(AA3)} \quad F_r a &\leq \neg F_s \neg a, \ r + s > 1 \\
\text{(AA4)} \quad F_r (a \land b) \land F_s (a \land \neg b) &\leq F_{r+s} a, \ r + s \leq 1 \\
\text{(AA5)} \quad \neg F_r (a \land b) \land \neg F_s (a \land \neg b) &\leq \neg F_{r+s} a, \ r + s \leq 1 \\
\text{(AA6)} \quad a &\leq b \Rightarrow F_r a \leq F_r b \\
\text{(AA7)} \quad \bigwedge_{r<s} F_{r_1 \cdots r_n} a &= F_{r_1 \cdots r_n} s a
\end{align*}

Table 1

The operator \( F_r \) is the algebraic counterpart of the logical modality \( L_r \). The first two axioms state tautologies, while the third captures the way \( F_r \) interacts with negation. Axioms (AA4) and (AA5) assert finite additivity, while (AA6) asserts monotonicity.

The most interesting axiom is the infinitary axiom (AA7). It asserts that \( F_{r_1 \cdots r_n} s a \) is the greatest lower bound of the set \( \{F_{r_1 \cdots r_n} r a \mid r < s\} \) with respect to the natural order \( \leq \). In SMPs, it implies countable additivity.

As expected, the formulas of Markovian logic modulo logical equivalence form a free countable Aumann algebra. Define \( \equiv \) on formulas in the usual way and let \([\varphi]\) denote the equivalence class of \( \varphi \) modulo \( \equiv \), and let \( \mathcal{L} = \{[\varphi] \mid \varphi \in \mathcal{L}\} \).

**Theorem 3.2** The structure

\[
(\mathcal{L}/\equiv, \rightarrow, [\bot], \{L_r\}_{r \in \mathbb{Q}_0}, \leq)
\]

is an Aumann algebra, where \([\varphi] \leq [\psi] \) iff \( \vdash \varphi \rightarrow \psi \).

A Stone–Markov process (SMP) is a Markov process \((M, \mathcal{A}, \theta)\), where \( \mathcal{A} \) is a distinguished countable base of clopen sets that is closed under the set-theoretic Boolean
operations and the operations
\[ F_r(A) = \{ m \mid \theta(m)(A) \geq r \}, \quad r \in \mathbb{Q}_0. \]

The measurable sets \( \Sigma \) are the Borel sets of the topology generated by \( A \). Morphisms of such spaces are required to preserve the distinguished base; thus a morphism \( f : \mathcal{M} \to \mathcal{N} \) is a continuous function such that
- for all \( m \in M \) and \( B \in \Sigma_N \),
  \[ \theta_M(m)(f^{-1}(B)) = \theta_N(f(m))(B); \]
- for all \( A \in \mathcal{A}_N \), \( f^{-1}(A) \in \mathcal{A}_M \).

Unlike Stone spaces, SMPs are not topologically compact, but we do postulate a completeness property that is a weak form of compactness, which we call saturation. One can saturate a given SMP by a completion procedure that is reminiscent of Stone–Čech compactification [15]. Intuitively, one adds points to the structure without changing the represented algebra. An MP is saturated if it is maximal with respect to this operation.

**Definition 3.3 [Stone–Markov Process]** A Markov process \( \mathcal{M} = (M, \mathcal{A}, \theta) \) with distinguished base is a *Stone–Markov process (SMP)* if it is saturated.

The morphisms of SMPs are just the morphisms of MPs with distinguished base as defined above.

The category of SMPs and SMP morphisms is denoted \( \text{SMP} \).

Fix an arbitrary countable Aumann algebra
\[ \mathcal{A} = (A, \rightarrow, \bot, \{ F_r \}_{r \in \mathbb{Q}_0}, \leq). \]

Let \( \mathcal{U}^* \) be the set of all ultrafilters of \( \mathcal{A} \). The classical Stone construction gives a Boolean algebra of sets isomorphic to \( \mathcal{A} \) with elements
\[ \langle a \rangle^* = \{ u \in \mathcal{U}^* \mid a \in u \}, \quad \langle \mathcal{A} \rangle^* = \{ \langle a \rangle^* \mid a \in A \}. \]

The sets \( \langle a \rangle^* \) generate a Stone topology \( \tau^* \) on \( \mathcal{U}^* \), and the \( \langle a \rangle^* \) are exactly the clopen sets of the topology.

Let \( \mathcal{F} \) be the set of elements of the form \( \alpha^r = F_{t_1 \ldots t_n} a \) for \( a \in A \) and \( t_1, \ldots, t_n, r \in \mathbb{Q}_0 \). As before, we consider this term as parameterized by \( r \); that is, if \( \alpha^r = F_{t_1 \ldots t_n} a \), then \( \alpha^s \) denotes \( F_{t_1 \ldots t_n} a \). The set \( \mathcal{F} \) is countable since \( A \) is. Axiom (AA7) asserts all infinitary conditions of the form
\[ \alpha^s = \bigwedge_{r < s} \alpha^r. \]
for $\alpha^s \in \mathcal{F}$. Let us call an ultrafilter $u$ bad if it violates one of these conditions in the sense that for some $\alpha^s \in \mathcal{F}$, $\alpha^r \in u$ for all $r < s$ but $\alpha^s \notin u$. Otherwise, $u$ is called good. Let $\mathcal{U}$ be the set of good ultrafilters of $\mathcal{A}$.

Let $\tau = \{ B \cap \mathcal{U} \mid B \in \tau^* \}$ be the subspace topology on $\mathcal{U}$, and let

$$\langle a \rangle = \{ u \in \mathcal{U} \mid a \in u \} = \langle a \rangle^* \cap \mathcal{U} \quad \langle A \rangle = \{ \langle a \rangle \mid a \in A \}.$$  

Then $\tau$ is countably generated by the sets $\langle a \rangle$ and all $\langle a \rangle$ are clopen in the subspace topology.

We can now form a Markov process $M(\mathcal{A}) = (\mathcal{U}, \Sigma, \theta)$, where $\Sigma$ is the $\sigma$-algebra generated by $\langle A \rangle$ and $\theta : \mathcal{U} \to \langle A \rangle \to [0, 1]$ is defined on the generators by

$$\theta(u)(\langle a \rangle) = \sup\{ r \in \mathbb{Q}_0 \mid F_r a \in u \} = \inf\{ r \in \mathbb{Q}_0 \mid \neg F_r a \in u \}.$$  

It can be shown that $\theta$ extends uniquely to a transition function [15].

**Theorem 3.4** If $\mathcal{A}$ is a countable Aumann algebra, then $M(\mathcal{A}) = (\mathcal{U}, \langle A \rangle, \theta)$ is a Stone Markov process.

Most of the technical difficulties of our earlier paper are in the proof of this theorem.

Let $\mathcal{M} = (M, \mathcal{B}, \theta)$ be a Stone Markov process with distinguished base $\mathcal{B}$. By definition, $\mathcal{B}$ is a field of clopen sets closed under the operations

$$F_r(A) = \{ m \in M \mid \theta(m)(A) \geq r \}.$$  

**Theorem 3.5** The structure $\mathcal{B}$ with the set-theoretic Boolean operations and the operations $F_r$, $r \in \mathbb{Q}_0$ is a countable Aumann algebra.

We denote this algebra by $\mathbb{A}(\mathcal{M})$. Now we have the duality theorem.

**Theorem 3.6 (Duality Theorem [15])**

(i) Any countable Aumann algebra $\mathcal{A}$ is isomorphic to $\mathbb{A}(M(\mathcal{A}))$ via the map $\beta : \mathcal{A} \to \mathbb{A}(M(\mathcal{A}))$ defined by

$$\beta(a) = \{ u \in \text{supp}(M(\mathcal{A})) \mid a \in u \} = \langle a \rangle.$$  

(ii) Any Stone Markov process $\mathcal{M} = (M, \mathcal{A}, \theta)$ is homeomorphic to $M(\mathbb{A}(\mathcal{M}))$ via the map $\alpha : \mathcal{M} \to M(\mathbb{A}(\mathcal{M}))$ defined by

$$\alpha(m) = \{ A \in \mathcal{A} \mid m \in A \}.$$  

In [15], we also give a categorical version of this theorem with the contravariant functors between the two categories given explicitly.
4 Extending the Duality to Metrized Markov Processes

We add quantitative structure to both Markov processes and Aumann algebras. We prove an extended version of the representation theorem for metrized Markov processes versus metrized Aumann algebras. This theorem states that starting from an arbitrary metrized Markov process, we can extend the Aumann algebra constructed in the previous section to a metrized Aumann algebra that preserves the pseudometric and conversely. In other words the natural isomorphisms that arise in the duality of [15] will turn out to be isometries.

We equip Stone Markov processes with a pseudometric that measures distances between the states of the MP. The key condition that we impose is that for a particular state \( m \), the diameters of the clopens containing \( m \) converges to 0.

**Definition 4.1** [Metrized Markov process] A metrized Markov process is a tuple \((M, d)\), where \( M = (M, A, \theta) \) is a Stone Markov process with \( A \) its countable base of clopens and \( d : M \times M \rightarrow [0, 1] \) is a pseudometric on \( M \) satisfying for arbitrary \( m \in M \) the property

\[
(M) \inf_{c \in A, m \in c} \sup \{d(n, n') \mid n, n' \in c\} = 0.
\]

The following lemma gives a number of conditions equivalent to \((M)\). In particular, it shows the connection between the topology of the Stone Markov space and the pseudometric topology.

**Lemma 4.2** For a metrized MP \((M, d)\), where \( M = (M, A, \theta) \), the following are equivalent:

(i) \( \forall m, \inf_{c \in A, m \in c} \sup \{d(n, n') \mid n, n' \in c\} = 0 \)

(ii) \( \forall m, m' \inf_{c \in A, m, m' \in c} \sup \{d(n, n') \mid n, n' \in c\} = d(m, m') \)

(iii) \( \forall m, \forall \varepsilon > 0 \exists c \in A \ (m \in c \land \forall n, n' \in c \ d(n, n') < \varepsilon) \)

(iv) \( \forall m \forall \varepsilon > 0 \exists c \in A \ (m \in c \land \forall n \in c \ d(m, n) < \varepsilon) \)

(v) The topology generated by \( A \) refines the pseudometric topology generated by \( d \).

(vi) The pseudometric \( d \) is continuous in both arguments with respect to the \( A \)-topology.

**Proof.** Note that (i) is just \((M)\).

(i) \( \Leftrightarrow \) (iii) is immediate from the definitions.

For (iii) \( \Rightarrow \) (iv), we can substitute \( m, n \) for \( n, n' \) in (iii).
To show (i) implies (ii), let \( y \). Then for all \( n \in c \), \( d(m,n) < \varepsilon/2 \) and \( d(n,m) < \varepsilon/2 \). Then for any \( n,n' \in c \), \( d(n,n') \leq d(n,m) + d(m,n') < \varepsilon/2 + \varepsilon/2 = \varepsilon \).

For (iv) \( \Leftrightarrow (v) \), let \( N_\varepsilon(m) = \{ x \mid d(m,x) < \varepsilon \} \). Then,

\[
\forall m \forall \varepsilon > 0 \exists c \in A \ (m \in c \land \forall n \in c \ d(m,n) < \varepsilon \land d(n,m) < \varepsilon) \\
\Leftrightarrow \forall m \forall \varepsilon > 0 \exists c \in A \ (m \in c \land \forall n \in c \ n \in N_\varepsilon(m)) \\
\Leftrightarrow \forall m \forall \varepsilon > 0 \exists c \in A \ (m \in c \land c \subseteq N_\varepsilon(m)).
\]

The last statement says that every basic open neighborhood of the pseudometric topology contains a basic open neighborhood of the \( A \)-topology, which says exactly that the \( A \)-topology refines the pseudometric topology.

For (iii) \( \Rightarrow (vi) \), to show that \( d \) is continuous in its second argument, let \( m,x \) and \( \varepsilon > 0 \) be arbitrary and let \( c \in A \) such that \( x \in c \) and for all \( n,n' \in c \), \( d(n,n') < \varepsilon \). Then for all \( y \in c \),

\[
d(m,y) \leq d(m,x) + d(x,y) < d(m,x) + \varepsilon \\
d(m,x) \leq d(m,y) + d(y,x) < d(m,y) + \varepsilon
\]

so \( d(m,y) \in (d(m,x) - \varepsilon, d(m,x) + \varepsilon) \). That \( d \) is continuous in its first argument follows from symmetry.

For the other direction, suppose \( d \) is continuous in its second argument. Then for all \( m \) and \( \varepsilon > 0 \), the set \( N_\varepsilon(m) \) is open and contains \( m \), thus there exists a basic open set \( c \in A \) such that \( m \in c \) and \( c \subseteq N_\varepsilon(m) \). Thus the \( A \)-topology refines the pseudometric topology of \( d \).

Statement (ii) implies (i) immediately by taking \( m' = m \) in (ii).

To show (i) implies (ii), let \( m, m' \) and \( \varepsilon > 0 \) be arbitrary. From (iv), we have \( c \in A \) such that \( m \in c \) and from (iii) we have that for all \( n,n' \in c \), \( d(n,n') < \varepsilon/2 \) and we have \( c' \in A \) such that \( m' \in c' \) and for all \( n,n' \in c' \), \( d(n,n') < \varepsilon/2 \). We claim that for all \( n,n' \in c \cup c' \), \( d(n,n') < d(m,m') + \varepsilon \), which will establish (ii).

If \( n,n' \in c \), then

\[
d(n,n') \leq d(n,m) + d(m,m') < \varepsilon/2 + \varepsilon/2 = \varepsilon \leq d(m,m') + \varepsilon.
\]

If \( n,n' \in c' \), the argument is the same, replacing \( m \) by \( m' \). If \( n \in c \) and \( n' \in c' \), then

\[
d(n',n) = d(n,n') \leq d(n,m) + d(m,m') + d(m',n') < \varepsilon/2 + d(m,m') + \varepsilon/2 = d(m,m') + \varepsilon.
\]

\( \Box \)
Definition 4.3 [Isometric Markov processes] Given two metrized MPs $(\mathcal{M}_i, d_i)$, where $\mathcal{M}_i = (\mathcal{M}_i, \Sigma_i, \theta_i)$ for $i = 1, 2$, an isometry from $\mathcal{M}_1$ to $\mathcal{M}_2$ is a map $f : \mathcal{M}_1 \to \mathcal{M}_2$ such that for arbitrary $m, n \in \mathcal{M}_1$,

$$d_1(m, n) = d_2(f(m), f(n)).$$

Now we introduce the metrized Aumann algebras. Despite their name, the metrized AAs are not directly equipped with a pseudometric structure, but with a concept of metric diameter. Later we will prove that the metric diameter can indeed define a pseudometric.

Definition 4.4 [Metrized Aumann algebra] A metrized Aumann algebra is a tuple $(\mathcal{A}, | |)$, where $\mathcal{A} = (\mathcal{A}, \rightarrow, \bot, \{ F_r \}_{r \in \mathbb{Q}_0}, \leq)$ is an Aumann algebra and $| | : \mathcal{A} \to [0, 1]$ is a metric diameter on $\mathcal{A}$, which is a map satisfying, for arbitrary $a, b \in \mathcal{A}$ and ultrafilter $\mathcal{U}$, the following properties

(A1) $| \bot | = 0$;

(A2) if $a \leq b$, then $|a| \leq |b|$;

(A3) if $a \land b \neq \bot$, then $|a \lor b| \leq |a| + |b|$;

(A4) $\inf \{|a| \mid a \in \mathcal{U}\} = 0$.

Definition 4.5 [Isometric Aumann Algebras] Given two metrizable Aumann Algebras $(\mathcal{A}_i, | |_i)$ for $i = 1, 2$, an isometry from $\mathcal{A}_1$ to $\mathcal{A}_2$ is a map $f : \mathcal{A}_1 \to \mathcal{A}_2$ such that for any $a \in \mathcal{A}_1$,

$$|a|_1 = |f(a)|_2.$$

We can now extend the duality results presented in the previous section to include the metric structure.

Consider a metrizable MP $(\mathcal{M}, d)$, where $\mathcal{M} = (\mathcal{M}, \mathcal{A}, \theta)$. As before, let $\mathcal{A}(\mathcal{M})$ be the AA constructed from $\mathcal{M}$. We extend this construction so that $\mathcal{A}(\mathcal{M})$ becomes a metrized AA. For arbitrary $a \in \mathcal{A}$, let

$$|a|_d = \sup \{d(m, n) \mid m, n \in a\},$$

under the assumption that $\sup \emptyset = 0$.

Lemma 4.6 $(\mathcal{A}(\mathcal{M}), | |_d)$ is a metrized Aumann Algebra.

Proof. (A1) $| \bot |_d = 0$ follows from the assumption that $\sup \emptyset = 0$.

(A2) If $c_1 \subseteq c_2$, then $\sup \{d(m, n) \mid m, n \in c_1\} \leq \sup \{d(m, n) \mid m, n \in c_2\}$.

(A3) Suppose that $c_1 \cap c_2 \neq \emptyset$. Let $z \in c_1 \cap c_2$, $x \in c_1$ and $y \in c_2$. Then, the triangle inequality for $d$ guarantees that

$$d(x, y) \leq d(x, z) + d(z, y) \leq \sup \{d(x, z) \mid x, z \in c_1\} + \sup \{d(z, y) \mid z, y \in c_2\} = |c_1| + |c_2|.$$
Consequently, \( \sup\{d(x, y) \mid x \in c_1, y \in c_2\} \leq |c_1| + |c_2| \) and similarly, 
\( \sup\{d(y, x) \mid x \in c_1, y \in c_2\} \leq |c_1| + |c_2| \). Since
\[
|c_1 \cup c_2| = \max\{|c_1|, |c_2|, \sup\{d(x, y) \mid x \in c_1, y \in c_2\}, \sup\{d(x, y) \mid x \in c_2, y \in c_1\}\},
\]
\[
|c_1 \cup c_2| \leq |c_1| + |c_2|.
\]

(A4) Using the notation of the previous duality theorem 3.6, we know that for any ultrafilter \( u \) of \( A(M) \) and any clopen \( c \in A \), we have that
\[
\alpha^{-1}(u) \in c \text{ iff } c \in u.
\]

Then, \( \inf\{|c|_d \mid c \in u\} = \inf\{|c|_d \mid \alpha^{-1}(u) \in c\} \).

Since \( \alpha \) is a bijection, \( \alpha^{-1}(u) \in M \) and \( |c|_d = \sup\{d(n, n') \mid n, n' \in c\} \), using (M) we obtain
\[
\inf_{c \in A, c \ni \alpha^{-1}(u)} \sup\{d(n, n') \mid n, n' \in c\} = 0,
\]
therefore \( \inf\{|c|_d \mid c \in u\} = 0 \). \( \square \)

Consider a metrizable AA \( (A, |\,|) \) and, as before, let \( \mathbb{M}(A) \) be the MP constructed from \( A \). We extend this construction so that \( \mathbb{M}(A) \) will become a metrizable MP.

For arbitrary ultrafilters \( u, v \) of \( A \), let
\[
\delta|\,|(u, v) = \inf\{|a| \mid a \in u \cap v\}.
\]

Lemma 4.7 \( (\mathbb{M}(A), \delta|\,|) \) is a metrized MP.

**Proof.** First, we prove that \( \delta|\,| \) is a pseudometric over the space of ultrafilters. From (A4) we can simply infer that \( \delta|\,|(u, u) = 0 \), while the symmetry of \( \delta|\,| \) follows from the definition.

We now prove the triangle inequality: let \( u, v, w \) be three arbitrary ultrafilters. Let \( a \in u \cap v, b \in u \cap w \) and \( c \in w \cap v \). Obviously \( b \cup c \in u \cap v \). Then,
\[
\inf_{a \in u \cap w} |a| \leq |b \cup c|.
\]

Since \( b \cap c \neq \emptyset \), using (A3) we get \( |b \cup c| \leq |b| + |c| \) which guarantees that for any \( b \in u \cap w \) and any \( c \in w \cap v \),
\[
\inf_{a \in u \cap w} |a| \leq |b| + |c|,
\]
implying
\[
\inf_{a \in u \cap w} |a| \leq \inf_{b \in u \cap w} |b| + \inf_{c \in w \cap v} |c|.
\]

Hence,
\[
\delta|\,|(u, v) \leq \delta|\,|(u, w) + \delta|\,|(w, v)
\]

225
which proves that \( \delta \) is a pseudometric.

It remains to verify (M). Since from theorem 3.6 we know that \( a \in u \) iff \( u \in \beta(a) \), (M) follows directly from (A4).

Finally, we extend the representation theorem 3.6 to include the metric structure. Essentially, we show that the isomorphisms \( \alpha \) and \( \beta \) of the duality theorem are isometries.

**Theorem 4.8 (The metric duality theorem)**

(i) Any metrizable countable Aumann algebra \((A, | \ |)\) is isomorphic to \((\mathcal{M}(A), | \delta_1 |)\) via the map \( \beta : A \to \mathcal{M}(A) \) defined by

\[
\beta(a) = \{ u \in \text{supp}(\mathcal{M}(A)) \mid a \in u \} = \{ a \}.
\]

Moreover, \( \beta \) is an isometry of metrizable Aumann algebras, i.e., for arbitrary \( a \in A \),

\[
|a| = |\beta(a)|_{\delta_1}.
\]

(ii) Any metrizable Stone Markov process \((M, d)\), where \( M = (M, A, \theta) \) is homeomorphic to \((\mathcal{M}(A), | \delta_1 |_d)\) via the map \( \alpha : M \to \mathcal{M}(A) \) defined by

\[
\alpha(m) = \{ A \in A \mid m \in A \}.
\]

Moreover, \( \alpha \) is an isometry of MPs, i.e., for arbitrary \( m, n \in M \),

\[
d(m, n) = \delta_1 |_d(\alpha(m), \alpha(n)).
\]

**Proof.** We only need to prove the two isometries.

(i). The isometry of AAs. We need to prove that \( |a| = |\beta(a)|_{\delta_1} \).

Observe that

\[
|\beta(a)|_{\delta_1} = \sup_{u,v \in \beta(a)} |(u,v)| = \sup_{u,v \in \beta(a)} \inf_{a' \in u \cap v} |a'|.
\]

Since \( \beta(a) \) is the set of all ultrafilters containing \( a \), \( a' \) quantifies over all elements that belong to the intersection of all ultrafilters containing \( a \). But this intersection is nothing else but the principal filter \( \uparrow a \) of \( a \). Hence, the previous equality became

\[
|\beta(a)|_{\delta_1} = \inf_{a' \in \uparrow a} |a'|.
\]

Now the monotonicity stated by (A2) guarantees that

\[
\inf_{a' \in \uparrow a} |a'| = |a|.
\]

(ii). The isometries of MPs. We need to prove that \( d(m, n) = \delta_1 |_d(\alpha(m), \alpha(n)) \).
From Lemma 4.2(ii) we know that
\[ d(m,n) = \inf_{c \in A, c \ni m,n} |c|_d. \]

From Theorem 3.6 we also know that
\[ m,n \in c \iff c \in \alpha(m) \cap \alpha(n). \]

Consequently,
\[ d(m,n) = \inf\{|c|_d \mid c \in \alpha(m) \cap \alpha(n)\} = \delta_{|d}(\alpha(m), \alpha(n)). \]

We have claimed earlier that the metric diameter on an Aumann algebra induces a pseudometric. We now demonstrate this.

Let \((A, \|\|)\) be a metrized AA. For arbitrary \(a, b \in A\) and \(\varepsilon > 0\), let
\[ B_\varepsilon(b) = \bigcup\{\beta(b') \mid b' \in A, |b'| \leq \varepsilon, b \wedge b' \neq \bot\}. \]

Intuitively, \(B_\varepsilon(b)\) is a ball that contains all ultrafilters that are at distance at most \(\varepsilon\) from some ultrafilter containing \(b\). This definition allows us to define a natural distance on \(A\) by
\[ d_{|\|}(a,b) = \inf\{\varepsilon > 0 \mid B_\varepsilon(b) \supseteq \beta(a) \text{ and } B_\varepsilon(a) \supseteq \beta(b)\}. \]

Intuitively, if in the light of the duality we think of the elements of \(A\) as sets of ultrafilters, then the previous distance is just the Hausdorff pseudometric of the distance between ultrafilters.

To prove that the previous construction is not void, we show in the next lemma that for any non-zero element \(a \in A\) the ball \(B_\varepsilon(a)\) is not empty for any \(\varepsilon\).

**Lemma 4.9** If \(a \neq \bot\), then for any \(\varepsilon > 0\) there exists \(a' \neq \bot\) such that \(a \wedge a' \neq \bot\) and \(|a'| \leq \varepsilon\).

**Proof.** Since \(a \neq \bot\), there exists an ultrafilter \(u\) such that \(a \in u\). For any other \(a' \in u, a \cap a' \in u\), hence \(a \cap a' \neq \bot\). Moreover, using (A4) there exists \(a' \in u\) such that \(|a'| \leq \varepsilon\). \(\square\)

Now we prove that \(d_{|\|}\) is indeed a pseudometric: it is the Hausdorff pseudometric of the pseudometric \(\delta_{|\|}\) on ultrafilters.

**Lemma 4.10** The function \(d_{|\|}\) previously defined on the support set of a metrizable Aumann Algebra is a pseudometric. Moreover,
\[ d_{|\|}(a,b) = \max\{\sup_{u \in \beta(a)} \inf_{v \in \beta(b)} \delta_{|\|}(u,v), \sup_{u \in \beta(a)} \inf_{v \in \beta(b)} \delta_{|\|}(v,u)\}. \]
We omit the proof, which is not completely trivial, from this abstract.

4.1 Metric Duality in Categorical Form

We present the previous results in a more categorical format. The categories of metrized Aumann algebras (MAA) and metrized Markov processes (MMP) are defined as follows.

The objects of MAA are metrized AAs and their morphisms are expansive morphisms of AAs, i.e., morphisms $f : A_1 \to A_2$ of AAs such that for any $a \in A_1$,

$$|a|_1 \leq |f(a)|_2.$$

The objects of MMP are metrized MPs and their morphisms are non-expansive morphisms of MPs, i.e., morphisms $f : M_1 \to M_2$ of SMPs such that for any $m,n \in M_1$,

$$d_1(m,n) \geq d_2(f(m), f(n)).$$

We define contravariant functors $\mathbb{A} : \text{MMP} \to \text{MAA}^{\text{op}}$ and $\mathbb{M} : \text{MAA} \to \text{MMP}^{\text{op}}$. The functor $\mathbb{A}$ on an object $\mathcal{M}$ produces the Aumann algebra $\mathbb{A}(\mathcal{M})$ defined in Theorem 3.5. On arrows $f : \mathcal{M} \to \mathcal{N}$ we define $\mathbb{A}(f) = f^{-1} : \mathbb{A}(\mathcal{N}) \to \mathbb{A}(\mathcal{M})$. We have previously proved that this is an Aumann algebra homomorphism. To see that it is also expansive, consider a morphism $f : \mathcal{M} \to \mathcal{N}$ such that for any $m,n \in \mathcal{M}$,

$$d_\mathcal{M}(m,n) \geq d_\mathcal{N}(f(m), f(n)).$$

Observe that for arbitrary $a \in \mathbb{A}(\mathcal{N})$,

$$|a|_{\mathbb{A}(\mathcal{N})} = |a|_{d_\mathcal{N}} \quad \text{and} \quad |f^{-1}(a)|_{\mathbb{A}(\mathcal{M})} = |f^{-1}(a)|_{d_\mathcal{M}}$$

and in this context the previous inequality guarantees that

$$|a|_{\mathbb{A}(\mathcal{N})} \leq |f^{-1}(a)|_{\mathbb{A}(\mathcal{M})}.$$

The functor $\mathbb{M} : \text{MAA} \to \text{MMP}^{\text{op}}$ on an object $\mathcal{A}$ gives the Stone–Markov process $\mathbb{M}(\mathcal{A})$ defined in Theorem 3.4. On morphisms $h : \mathcal{A} \to \mathcal{B}$, it maps ultrafilters to ultrafilters by $\mathbb{M}(h) = h^{-1} : \mathbb{M}(\mathcal{B}) \to \mathbb{M}(\mathcal{A})$; that is,

$$\mathbb{M}(h)(u) = h^{-1}(u) = \{ A \in \mathcal{A}_N \mid h(A) \in u \}.$$

Another way to view $\mathbb{M}(h)$ is by composition, recalling that an ultrafilter can be identified with a homomorphism $\mathfrak{u} : \mathcal{A} \to 2$ by $u = \{ a \mid \mathfrak{u}(a) = 1 \}$. In this view,

$$\mathbb{M}(h)(\mathfrak{u}) = \mathfrak{u} \circ h,$$

where $\circ$ denotes function composition.
We have proven in our previous paper [15] that this is a morphism of SMPs.

That it is also non-expansive can be demonstrated as follows.

Let $h : \mathcal{A} \to \mathcal{B}$ be a morphism such that for arbitrary $a \in \mathcal{A}$,

$$|a|_\mathcal{A} \leq |f(a)|_\mathcal{B}.$$ 

Using the previous results it is not difficult to verify that for arbitrary ultrafilters $u, v$ of $\mathcal{B}$,

$$\delta|_\mathcal{B}(u, v) \geq \delta|_\mathcal{A}(h^{-1}(u), h^{-1}(v)).$$

And since

$$\delta|_\mathcal{B}(u, v) = \delta_{\mathcal{M}(\mathcal{B})}(u, v) \text{ and } \delta|_\mathcal{A}(h^{-1}(u), h^{-1}(v)) = \delta_{\mathcal{M}(\mathcal{A})}(h^{-1}(u), h^{-1}(v)),$$

We obtain that $\mathcal{M}(h)$ is non-expansive.

**Theorem 4.11** The functors $\mathcal{M}$ and $h$ define a dual equivalence of categories.

\begin{center}
\begin{tikzcd}
\mathcal{A} \arrow{r}{\mathcal{M}} \arrow{r}[swap]{MMP} & \mathcal{M}^{\text{op}} \arrow{r}{\mathcal{A}}
\end{tikzcd}
\end{center}

5 Conclusions

We have extended the duality theory of [15] to a quantitative setting. It is important to note that the conditions we have imposed on the pseudometric relate the topology of the Markov process to the pseudometric topology. This can be seen from the fact that the pseudometric topology is refined by the Stone topology. We have defined our Stone Markov processes to be Hausdorff spaces, which was necessary for the duality theory. In effect, this means that the clopens separate points; in other words, one cannot have two states that satisfy exactly the same formulas. In view of the logical characterization of bisimulation, this implies that no two distinct states are bisimilar; that is, the process is already minimal with respect to bisimulation. If we look at a broader class of Markov processes, then we would have possibly nontrivial bisimulations on the space. The Stone Markov processes would be a reflective subcategory with the reflector sending each Markov process to a version of the process with all the bisimulation equivalence classes collapsed to point; this would be a Stone Markov process. How does the topology on a Stone Markov process “know” about the transition structure? Note that the base of clopens is required to be closed under the $F_r$ operations, which are defined in terms of the transition function.

The only work of which we are aware similar to this is a paper by Banaschewski and Pultr [2] called “A Stone duality for metric spaces.” They are not working with Markov processes, so there is nothing like the Aumann algebra structure there. They
Kozen, Mardare, and Panangaden gave us the idea of using a metric diameter, but their axiomatization is different and the proofs that we developed do not resemble theirs.

The main impact of this work is to put quantitative reasoning about Markov processes on a firmer footing. It has been over a decade since metric analogues of bisimulation were developed, but they have not had the impact that they might have had. One reason is that with ordinary logical reasoning, one has a clear understanding of what completeness means, thus users of these logics have a good understanding of the power of the principles they are using. What does completeness mean for metric reasoning and approximate reasoning in general? The standard Stone-type duality theorem captures the concept of completeness; it is our hope that the present work will pave the way towards a similar understanding of approximate reasoning principles. There is much to be done, however. In a previous paper [17] we began investigating the relationship between the logic and metrics on Markov processes. The results of the present paper could perhaps strengthen and deepen these preliminary results.

Acknowledgments

Panangaden’s research was supported by an NSERC grant. Mardare’s research was supported by VKR Center of Excellence MT-LAB and by the Sino-Danish Basic Research Center IDEA4CPS. Kozen’s research was supported by the National Security Agency.

References


Kozen, Mardare, and Panangaden


Canonicity of Weak $\omega$-Groupoid Laws Using Parametricity Theory

Marc Lasson

INRIA Paris-Rocquencourt, PiR2, Univ Paris Diderot, Sorbonne Paris Cité
F-78153 Le Chesnay France

Abstract

We show that terms witnessing a groupoid law from the $\omega$-groupoid structure of types are all propositionally equal. Our proof reduce this problem to the unicity of the canonical point in the $n$-th loop space and conclude using Bernardy’s parametricity theory for dependent types.

Keywords: type theory, parametricity, loop spaces, groupoids, identity types

1 Introduction

The synthetic approach to weak $\omega$-groupoids promoted by the univalent foundation program [8] is the idea that (homotopy) type theory should be the primitive language in which spaces, points, paths, homotopies are derived. Following this approach, spaces are represented by types, points by inhabitants and paths by equalities between points (also known as identity types) and algebraic properties of these objects should not be enforced a priori (for instance by axioms) but should be derived directly from the language. To justify the synthetic approach, one should prove the canonicity of each definition, in the sense that no important choice should be made by choosing a particular implementation of a definition over another.

Garner, van den Berg [9] and Lumsdsaine [5] independently showed that in type theory, each type can be equipped with a structure of weak $\omega$-groupoids. For this, they show that a minimal fragment of Martin-Löf type theory, where identity types are the only allowed type constructors, bears a weak $\omega$-category structure. Informally, these results state the possibility to express algebraic properties of weak $\omega$-groupoids as types and in each case to find a canonical inhabitant of these types reflecting the fact that the property holds. Identities, inversion and concatenation of
Lasson

path, associativities, involution of inversion, horizontal and vertical compositions of 2-paths, are all examples of groupoid laws. The canonicity of witnesses of groupoid laws here means there is a path between any two inhabitants of the law witnessing their equality, but also that this path should be canonical: there should be a path between any two paths between two inhabitants of the same law, and so on ... This canonicity is already a known fact within the fragment. The main result of this article is to extend the canonicity to the whole Martin-Löf type theory.

In this work, we follow a syntactic approach inspired by Brunerie [3] to formalize the notion of groupoid law. We call groupoid law any closed type \( \forall \Gamma.c \) such that the sequent \( \Gamma \vdash c : \text{Type} \) is derivable in the minimal fragment and such that the context \( \Gamma \) is contractible. A contractible context is a context of the following shape:

\[
A : \text{Type}, a : A, x_1 : C_1, y_1 : M_1 = x_1, \ldots, x_n : C_n, y_n : M_n = x_n \text{ where } x_i \text{ does not occur in } M_i.
\]

The shape of these contexts is stable by path-induction, which allows to find an inhabitant of any groupoid law by successive path inductions. We show that this inhabitant is canonical, even outside of the fragment.

The main idea of the proof is to use successive path inductions to reduce the problem of the uniqueness of inhabitants of a given groupoid law to the uniqueness of the canonical point inhabiting a parametric loop space. Given a base type \( A \) and a point \( a : A \), the \( n \)-th loop space and its canonical point are inductively defined by:

\[
\begin{align*}
\Omega_0(A, a) & := A \\
\Omega_{n+1}(A, a) & := \Omega_n(a = a, 1_a) \\
\omega_0(A, a) & := a \\
\omega_{n+1}(A, a) & := \omega_n(a = a, 1_a)
\end{align*}
\]

where \( 1_a : a = a \) denotes the reflexivity. Thus for any integer \( n \), \( \forall X : \text{Type}, x : X.\Omega_n(X, x) \) is a groupoid law inhabited by \( \lambda X : \text{Type}, x : X.\omega_n(X, x) \) (note that using one universe, it is possible to internalize the quantification over \( n \); everything that we state here will be true whether or not this is used). We call this groupoid law the \( n \)-th parametric loop space.

The 0-th parametric loop space, is the polymorphic type \( \forall X : \text{Type}.X \rightarrow X \) of identity functions, and its canonical inhabitant is \( \lambda X : \text{Type}, x : X.1_x \), i.e. the identity function. This term is the only one up to function extensionality inhabiting its type. The standard tool to prove this kind of properties is by using Reynold’s parametricity theory [7] which was introduced to study the behavior of type quantifications within polymorphic \( \lambda \)-calculus (a.k.a. System F). It refers to the concept that well-typed programs cannot inspect types; they must behave uniformly with respect to abstract types. Reynolds formalizes this notion by showing that polymorphic programs satisfy the so-called logical relations defined by induction on the structure of types. This tool has been extended by Bernardy et al. [2] to dependent type systems. It provides a uniform translation of terms, types and contexts preserving typing (the so-called abstraction theorem). In its unary version (the only needed for this work), logical relations are defined by associating to any well-formed type \( A : \text{Type} \) a predicate \( \llbracket A \rrbracket : A \rightarrow \text{Type} \) and to any inhabitant \( M : A \) a witness \( \llbracket M \rrbracket : \llbracket A \rrbracket M \) that the \( M \) satisfies the predicate. This translation may be extended to cope with identity types by setting \( \llbracket a = b \rrbracket : a = b \rightarrow \text{Type} \) to be the predicate \( \lambda p : a = b.p_\ast(\llbracket a \rrbracket) = [b] \) where \( p_\ast \) is the transport along \( p \) of the predicate gener-
ated by the common type of $a$ and $b$. Then, it is easy -although quite verbose- to find a translation of introduction and elimination rules of identity types as well as checking that these translations preserve computation rules. This allows to extend Bernardy’s abstraction theorem to identity types. Using this framework, we are able to generalize the uniqueness property of the polymorphic identity type to any parametric loop space. The proof proceed by induction on the index of the loop space and uses algebraic properties of transport.

Outline of the paper.

In Section 2, we introduce the type theoretical setting that is used in the article. Section 3 is devoted to the proof that Bernardy’s parametricity may be extended to cope with identity types. In Section 4, we use this translation to prove the canonicity result for loop spaces; we prove that all inhabitants of parametric loop spaces are propositionally equal (Theorem 4.3). In Section 5, we introduce the fragment MLID of type theory to define our notion of groupoid laws and we show that the result of Section 4 may be generalized to all groupoid laws (Theorem 5.6). Finally Section 6 is devoted to various discussions.

2 Presentation of the syntax

We give a presentation of Martin-Löf type theory with identity types and universes which is close to the syntax of pure type systems [1] in order to reuse the parametricity theory presented in [2]. In this framework, computation rules are treated in an untyped way and subtyping of universes is achieved using Luo’s cumulativity relation introduced for the extended calculus of constructions [6]. The reader may be more used to judgemental presentations of type theory where each computation steps are checked to be well-typed. This is just a matter of presentation; all the material presented here could be adapted without much effort to suit type systems using a judgemental equality. Also, the use of cumulativity is not really needed but makes our results more general and closer to implementations such as coq.

The terms of the system are given by the following grammar:

$$
A, B, C, M, N, U, V := \ x \ | \ (M \ N) \ | \ \lambda x : A. M \ | \ \forall x : A. B \ | \ \text{Type}_i \\
\ | \ M =_A N \ | \ 1^C_M \ | \ J_{x : C, y : M =_A (B, U, V)}
$$

where universes $\text{Type}_i$ are indexed by $i \in \mathbb{N}$. Variables are considered up to $\alpha$-conversion and we write $M[N/x]$ to denote the term obtained by substituting all free occurrences of $x$ in $M$ by $N$. To ease the reading of terms, we allow ourselves to omit some typing annotations when they could be guessed from the context (in particular $M = N$, $1^C_M$ and $J(B, U, V)$ will be used often in this text).

The grammar of terms is obtained by adding to the syntax of pure type systems:

- The type constructor $M =_A N$ for forming identity types,
- The introduction rule for identity types, $1^C_M$, to witness the reflexivity $M =_C M$,
The elimination rule (also known as “path-induction”) \( J_{x:C,y:M=x} A(B, U, V) \).

Given any dependent type \( A(x, y) \) which depends on a point \( x \) and a path \( y \) from a base point \( M \) to \( x \), given a witness for \( A(M, 1_M) \) (the base case of the induction), given a point \( U \) and a path \( V \), the path-induction provides an inhabitant of \( A(U, V) \). This formulation of path-induction, due to Paulin-Möhring (also called based path induction), is equivalent to the other version where \( A \) is parametrized by two points and a path between them.

We use the symbol \( \equiv \) to denote the syntactic equality (up to \( \alpha \)-conversion) between terms. The conversion between terms will be denoted by \( M \equiv_\beta N \), it is defined as the smallest congruence containing the usual \( \beta \)-reduction \( (\lambda x : A.M) N \equiv_\beta M[N/x] \) and the computation rule for identity types: \( J_{x:C,y:M=x} A(B, U, V) \equiv_\beta N \). And the cumulativity order is defined as the smallest partial order \( \preceq \) compatible with \( \equiv_\beta \) and satisfying for \( i \leq j \):

\[
\forall x_1 : A_1, ..., x_n : A_n. \text{Type}_i \preceq \forall x_1 : A_1, ..., x_n : A_n. \text{Type}_j
\]

Like in the extended calculus of constructions, the cumulativity rule is not fully contravariant with respect to the domain of functions (\( A' \preceq A \) and \( B \preceq B' \) does not imply \( \forall x : A.B \preceq \forall x : A'.B' \)) otherwise it would break the decidability of type checking. Contexts are finite lists of the form \( x_1 : A_1, \ldots, x_n : A_n \) mapping a variable to its type. The rules of the type system are given in Figure 1. As we follow standard lines, we do not develop in details the metatheory of the system.

The non-dependent version of path-induction is called transport and is defined by

\[
P_{* : C,X} (M) \equiv J_{y:M=x} X(M, V, P)
\]

where \( y \) does not occur in \( X \). It is often used to coerce between type families: given a type family \( X : C \to \text{Type} \), a path \( P : U = V \) between two points \( U, V : C \), and a term \( M \) in \( X_U \), the transport \( P_{* : C,X} (M) \) of \( M \) along \( P \) inhabits \( X_V \). It satisfies the following derivable rule:

\[
\frac{\Gamma, x : C \vdash X : \text{Type} \quad \Gamma \vdash P : U =_C V \quad \Gamma \vdash M : X[U/x] \quad \Gamma \vdash P_{* : C,X} (M) : X[V/x]}{\Gamma \vdash \text{Transport}}
\]

We sometimes also omit the type family when it can be guessed from the context. The computation rule tells us that transporting along a reflexivity is same as doing nothing: \( 1_{U,*}(M) \equiv_\beta M \).

3 Extending Relational Parametricity to Identity types

In this section, we explain how to extend Bernardy’s parametricity translation below to primitive identity types. Then we prove that this extension preserves typing (The-
orem 3.2).

where \( M \in [A] \) simply stands as a notation for \([A] M\) (it makes formulas a bit easier to read). As said in the introduction, the predicate generated by an identity type \( M =_C N \) is defined by the type family over \( M =_C N \) selecting paths transporting \( M \) to \( N \):

\[
[M =_C N] : M =_C N \to \text{Type}
\]

Thanks to the computational rule

\[
1^C_M \in [M =_C M] \equiv_\beta 1^C_M \ xC.x \in [C] ([M]) =_{M \in [C]} [M] \equiv_\beta [M] =_{M \in [C]} [M]
\]

the translation \([1^C_M] : 1^C_M \in [M =_C M]\) of reflexivity is a reflexivity: \([1^C_M] \equiv 1^C_M =_{M \in [C]} [M]\). And finally, the elimination rule is translated in terms of nested path-
The translation of the predicate of path inductions are quite verbose and make the translation hard to read. However if we ignore the annotation, we see that the translation \([ J(B, U, V) ] \) is simply a duplication of path induction which should be compared to the duplication in abstractions and applications. We can check that this translation behaves well with respect to substitution and conversion.

**Lemma 3.1 (Substitution and conversion lemma)** We have:

1. \([ M[N/x] ] \equiv [ M][N/x, [ N] / x_R] \),
2. If \( M \equiv_\beta M' \), then \([ M] \equiv_\beta [ M'] \).
3. If \( M \not\approx M' \), then \([ M] \not\approx [ M'] \).

**Proof.** The proof of (i) is a routine proof by induction on \( M \). And (iii) is a rather direct consequence of (ii). To prove (ii), we only do here the only check that is not in Bernardy’s translation:

\[
[ J(N, M, 1^C_M) ] \equiv J(J([N], M, 1^C_M), [M], [1^C_M]) \\
\equiv J(J([N], M, 1^C_M), [M], 1^{M\in[C]}_M) \\
\equiv [N]
\]

Now we can check that this extension preserve typing:

**Theorem 3.2 (Abstraction)** If \( \Gamma \vdash M : A \), then

\[
\begin{cases}
[ \Gamma ] \vdash M : A & (a) \\
[ \Gamma ] \vdash [M] : M \in [A] & (b)
\end{cases}
\]

moreover if \( \Gamma \) wf then \([ \Gamma ] \) wf (c).

**Proof.** The abstraction theorem is proved by induction on derivations. In each cases, proving the statement (a) is straightforward. Since the treatment of other rules is now standard, we only deal here with the rules concerning identity types. Even though we do not detailed it here, the treatment of CONVERSION uses previous lemma.

- **IDENTITY:** The induction hypothesis gives us \([ \Gamma ] \vdash [M] : M \in [C] \) (1), \([ \Gamma ] \vdash [N] : N \in [C] \) (2) and \([ \Gamma ], x : C \vdash x \in [C] : \text{Type} \) (3). Using \text{TRANSPORT} we derive from (1) and (3), that \([ \Gamma ], p : M =_C N \vdash p : \lambda x : C.x \in C ([M]) : N \in [C] \). Then, using \text{IDENTITY} and (2) we build a derivation for \([ \Gamma ], p : M =_C N \vdash \]

\( Lasson \)
Let $T$ which means, using (1) we can check that $\Gamma \vdash [M =_C N] : M =_C N \to \text{Type}_i$ with ABSTRACTION.

- **REFLEXIVITY:** By induction hypothesis, we have $[\Gamma] \vdash [M] : M \in \mathbb{C}$ (1). Using (1) we can check that $[\Gamma] \vdash 1^{M \in \mathbb{C}}_{[M]} : [M] =_{\mathbb{C}} [M]$ which is convertible to $[\Gamma] \vdash 1^{M \in \mathbb{C}}_{[M]} : 1^C_{M\times([M])} =_{M\subseteq\mathbb{C}} [M]$. So, we conclude $[\Gamma] \vdash 1^C_{M} : 1^C_{M} \in \mathbb{C}$.

- **PATH-INDUCTION:** The induction hypothesis for (b) are:

\[
\begin{align*}
[\Gamma] & \vdash [M] : M \in \mathbb{C} \quad (1) \\
[\Gamma, x : C, y : M = x] & \vdash [P] : P \to \text{Type}_i \quad (2) \\
[\Gamma] & \vdash [B] : B \in [P[M/x, 1^C_{M/y}]] \quad (3) \\
[\Gamma] & \vdash [U] : U \in \mathbb{C} \quad (4) \\
[\Gamma] & \vdash [V] : V \in \mathbb{C} = U \quad (5)
\end{align*}
\]

and the induction hypothesis for (a) are:

\[
\begin{align*}
[\Gamma] & \vdash M : C \quad (6) \\
[\Gamma, x : C, y : M = x] & \vdash P : \text{Type}_i \quad (7) \\
[\Gamma] & \vdash B : P[M/x, 1^C_{M/y}] \quad (8) \\
[\Gamma] & \vdash U : C \quad (9) \\
[\Gamma] & \vdash V : M = U \quad (10)
\end{align*}
\]

Let $T$ be the following term:

\[
J_{\forall x:C,y:M=x,J(B,x,y)\in[P][y_*(M)]/x,1_{y_*(M)}]/y}(B,U,V)
\]

First, we have to typecheck $T$ by showing that

\[
[\Gamma] \vdash T : J(B, U, V) \in [P][y_*(M)]/x,1_{y_*(M)}]/yR, U/x, V/y \quad (*)
\]

which means, using PATH-INDUCTION, checking:

- The predicate is well-formed:

\[
[\Gamma], x : C, y : M = x \vdash J(B, x, y) \in [P][y_*(M)]/x,1_{y_*(M)}]/yR, U/x, V/y : \text{Type}
\]

This is obtained by substituting $x_R$ and $y_R$ in (2) using a derivation of

\[
[\Gamma], x : C, y : M = x \vdash y_*(M) : x \in \mathbb{C}
\]

and of $[\Gamma], x : C, y : M = x \vdash 1_{y_*(M)} : y_*(M) = y_*(M)$ which are easily derived from (1) and by applying $[\Gamma], x : C, y : M = x \vdash J(B, x, y) : P$ to the substituted derivation.

- The arguments are correct : We derive

\[
[\Gamma] \vdash [B] : J(B, M, 1^C_M) \in [P][y_*(M)]/x,1_{y_*(M)}]/yR, M/x, 1^C_M/y
\]

by noticing using computation rules that:

\[
J(B, M, 1^C_M) \in [P][y_*(M)]/x,1_{y_*(M)}]/yR, M/x, 1^C_M/y \equiv_{\beta} B \in [P][M/x, M]/x, 1^C_M/y, 1^C_M/y
\]

238
and therefore CONVERSION and (3) allow us to conclude. Finally, we have to check that target point and path are correct, i.e. \([\Gamma] \vdash U : C\) and \([\Gamma] \vdash V : M = U\) which are given by (9) and (10).

Let \(K\) be the following term:

\[
J_{xR:U\in\llbracket C\rrbracket,yR:V_{*}(\llbracket M\rrbracket)} = xR.J_{yz:C,y:z=M=x,P(B,U,V)\in\llbracket P[U/x,V/y]\rrbracket}(T,\llbracket U\rrbracket,\llbracket V\rrbracket)
\]

We now want to check that : \([\Gamma] \vdash K : J_{yz:C,y:z=M=x,P(B,U,V)\in\llbracket P[U/x,V/y]\rrbracket}\) using PATH-INDUCTION, we have to show that :

1. The predicate is well-formed:

\[
[\Gamma], xR : U \in \llbracket C\rrbracket, yR : V_{*}(\llbracket M\rrbracket) = xR \vdash J_{yz:C,y:z=M=x,P(B,U,V)\in\llbracket P[U/x,V/y]\rrbracket} : \text{Type}_{i}
\]

which is obtained by substituting \(x\) and \(y\) in (2) using (9) and (10) and by applying \([\Gamma] \vdash J_{yz:C,y:z=M=x,P(B,U,V)\in\llbracket P[U/x,V/y]\rrbracket}\) to the substituted derivation.

2. The arguments are correct : We use (*) for type checking \(T\) and we use (4) and (5) for type checking \(\llbracket U\rrbracket\) and \(\llbracket V\rrbracket\).

\[\square\]

## 4 Canonicity in parametric loop spaces

The goal of this section is to prove Theorem 4.3. The following lemma is needed to perform the main induction in Lemma 4.2. The proofs terms are described in a semi-formal style for reading purpose, they could be easily constructed from the prose. However, as a consequence of Theorem 4.3, the precise shape of these terms is not important. In this section, we use \(p \cdot q\) and \(p^{-1}\) to denote respectively the concatenation of paths and the inverse.

### Lemma 4.1

Let \(P : A \rightarrow \text{Type}\) be a type family and \(u : Pa\) for some \(a : A\) and assume we have \(\phi : \forall x : A.P\ x \rightarrow a = x\). Then, for all \(p : a = a\) such that \(p_{*}(u) = u\), we have \(1 = p\).

**Proof.** By applying the functorial action of path on \(p_{*}(u) = u\) using \(\phi_{a}\) we obtain a two dimensional path \(\phi_{a}(p_{*}(u)) = \phi_{a}(u)\) and by the naturality of transport (Lemma 2.3.11 in [8]), we obtain a path \(p_{*}(\phi_{a}(u)) = \phi_{a}(u)\). The left-hand side path \(p_{*}(\phi_{a}(u))\) is transport over an identity type, we therefore have \(p_{*}(\phi_{a}(u)) = \phi_{a}(u) \cdot p\) and therefore \(\phi_{a}(u) \cdot p = \phi_{a}(u)\). So, we conclude \(1 = p\) by cancelling by \(\phi_{a}(u)\) and by symmetry. \[\square\]

Here is the main induction which allows us to derive Theorem 4.3:

**Lemma 4.2** Let \(A, P, a, u\) and \(\phi\) defined as in previous lemma. Then for all \(n\)-dimensional loop \(p : \Omega_{n}(A,a)\), the type family \(\llbracket \Omega_{n}\rrbracket(A,P,a,u) : \Omega_{n}(A,a) \rightarrow \text{Type}\) satisfies : \(\forall p : \Omega_{n}(A,a),\llbracket \Omega_{n}\rrbracket(A,P,a,u) p \rightarrow \omega_{n}(A,a) = p\).

**Proof.** By induction on \(n\), the base case is exactly witness by \(\phi\). For the inductive step, we need to prove \(\forall p : \Omega_{n+1}(A,a), p \in \llbracket \Omega_{n+1}\rrbracket(A,P,a,u) \rightarrow \omega_{n+1}(A,a) = p\)
which is convertible to:

\[ \forall p : \Omega_n(a = a, 1).p \in [\Omega_n](a = a, \lambda q : a = a.q_u(u) = u, 1, 1) \rightarrow \omega_n(a = a, 1) = p \]

We may therefore apply the induction hypothesis by providing a proof that \( \forall q : a = a.q_u(u) = u \rightarrow 1 = q \) which is given by previous lemma.

We can deduce:

**Theorem 4.3 (Canonicity for loop spaces)** If \( \vdash M : \forall A : \text{Type}, a : A.\Omega_n(A, a) \) then there is a term \( \pi \) such that \( \vdash \pi : \forall A : \text{Type}, a : A.\omega_n(A, a) = MAa \).

**Proof.** The abstraction theorem gives us a proof \([M] \) that \( \forall A : \text{Type}, A_R : A \rightarrow Type, a : A, a_R : A_RA_R a.(MAa) \in [\Omega_n](A, A_R, a, a_r) \) by instantiating \( A_R \) with \( \lambda x : A.a = x \) we can conclude by applying previous lemma.

Note that as a corollary, the proof \( \pi \) is unique up to propositional equality (and so on). If we have two such proofs \( \pi \) and \( \pi' \) then \( (\pi A a) \cdot (\pi' A a)^{-1} \) is of type \( \omega_n(A, a) = \omega_{n+1}(A, a) \). Therefore, by applying the previous theorem we obtain a proof of \( (\pi A a) \cdot (\pi' A a)^{-1} = \omega_{n+1}(A, a) \) which is also \( (\pi A a) \cdot (\pi' A a)^{-1} = 1_{\omega_n(A, a)} \).

Therefore we conclude \( (\pi A a) = (\pi' A a) \).

## 5 Groupoid laws

In this section, we start by describing a fragment MLID of the previous type system MLTT. This sub-system is used to characterise groupoid laws. Informally, it is obtained from MLTT by removing the rules ABSTRACTION, APPLICATION and UNIVERSES and by restricting valid sequents to the contractible contexts defined in the introduction. In the absence of function spaces, the rule PATH-INDUCTION has to be strengthened in order to be able to make a path induction along a path which is not the last one of the context. Therefore, we need to extend the grammar of terms in MLID with terms of the shape \( \mathbf{J}_{y : C, y : M = x, \Delta, P}(B, U, V, \overrightarrow{W}) \) where \( \Delta \) is a context and where the vectorial notation \( \overrightarrow{W} \) denotes a tuple \( (W_1, ..., W_n) \) of terms.

The typing rules are given in Figure 2. Formally a context \( \Gamma \) is said to be contractible if \( \Gamma \text{ cont} \) is derivable; the reader should notice that all contexts occurring in derivations of MLID are contractible. In the typing rules, we write \( \Gamma \vdash \overrightarrow{W} : \Delta \) to denote the conjunction for \( k = 1, \cdots, n \) of \( \Gamma \vdash W_k[A_1/y_1, \cdots, A_k-1/y_{k-1}] : A_k \) when \( \Delta \) is of the shape \( y_1 : A_1, \cdots, y_n : A_n \). Moreover, \( \overrightarrow{W}/\Delta \) denotes the iterated substitution \( [W_1/y_1, \cdots, W_n[A_1/y_1, \cdots, A_n-1/y_{n-1}]/y_n] \).

In order to embed MLID into MLTT, we translate extended path inductions into normal ones according to the following translation:

\[
\mathbf{J}_{y : C, y : M = x, \Delta, P}(B, U, V, \overrightarrow{W}) \equiv \\
\lambda \overrightarrow{x} : \Delta. (\mathbf{J}_{y : C, y : M = x, \forall \Delta, P}(\lambda x : \Delta[M/x, 1_M/y], B, U, V) \overrightarrow{x})
\]

where \( \overrightarrow{y} : \Delta \) means that \( y_1, \cdots, y_n \) are the variables assigned in \( \Delta \), and where \( \forall \Delta, \lambda \Delta, \) and \( (J \overrightarrow{x}) \) denote respectively iterated products, abstractions and appli-
Lasson

<table>
<thead>
<tr>
<th>( A : \text{Type}_0, x : A )</th>
<th>( \Gamma \text{ contr} )</th>
<th>( \Gamma \vdash_{\text{id}} B : \text{Type}_0 )</th>
<th>( \Gamma \vdash_{\text{id}} M : B )</th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT</td>
<td>( \Gamma, x : B, p : M =_B x \text{ contr} )</td>
<td>CONTR</td>
<td></td>
</tr>
<tr>
<td>( \Gamma \vdash_{\text{id}} M : C )</td>
<td>( \Gamma \vdash_{\text{id}} N : C )</td>
<td>( \Gamma \vdash_{\text{id}} C : \text{Type}_0 )</td>
<td>( \Gamma \vdash_{\text{id}} M : C )</td>
</tr>
<tr>
<td>( \Gamma \vdash_{\text{id}} M =_C N : \text{IDENTITY} )</td>
<td>( \Gamma \vdash_{\text{id}} M : \text{Type}_0 )</td>
<td>REFLEXIVITY</td>
<td></td>
</tr>
<tr>
<td>( \Gamma \text{ contr} )</td>
<td>( (x : A) \in \Gamma )</td>
<td>( \Gamma \vdash_{\text{id}} M : A' )</td>
<td>( \Gamma \vdash_{\text{id}} A' : \text{Type}_0 )</td>
</tr>
<tr>
<td>VARIABLE</td>
<td>( \Gamma \vdash_{\text{id}} M : A )</td>
<td>A' ( \equiv_{\beta} A )</td>
<td></td>
</tr>
<tr>
<td>( \Gamma \vdash_{\text{id}} M : C )</td>
<td>( \Gamma \vdash_{\text{id}} U : C )</td>
<td>( \Gamma \vdash_{\text{id}} V : M = U )</td>
<td></td>
</tr>
<tr>
<td>( \Gamma, x : C, y : M = x, \Delta \vdash_{\text{id}} P : \text{Type}_0 )</td>
<td>( \Gamma \vdash_{\text{id}} \overrightarrow{W} : \Delta )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \Gamma, \Delta[M/x, \overrightarrow{1}<em>{M/y}] \vdash</em>{\text{id}} B : P[M/x, \overrightarrow{1}_{M/y}] )</td>
<td>EXTENDED-PATH-INDUCTION</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 2. The minimal fragment MLID of type theory with identity types.

cations. It is then straightforward to check that EXTENDED-PATH-INDUCTION is an admissible rule of MLTT.

Groupoid laws are characterized by a contractible context and a derivable type in MLID (Figure 3 contains some examples of groupoid laws):

**Definition 5.1** [Groupoid law] A groupoid law is a term of the shape \( \forall \Gamma.C \) such that \( \Gamma \vdash_{\text{id}} C : \text{Type}_0 \).

The only groupoid laws in the “initial” contractible context \( A : \text{Type}, a : A \) are loop spaces. Since we do not change the base type and the base point of loop spaces in this section we simply denote by \( \omega_n \) (resp. \( \Omega_n \)) the terms \( \omega_n(A, a) \) (resp. \( \Omega_n(A, a) \)). Using these notations we have:

**Lemma 5.2** If \( A : \text{Type}, a : A \vdash_{\text{id}} T : \text{Type}_0 \), then

(i) there exists \( |T| \in \mathbb{N} \) such that \( T \equiv_{\beta} \Omega_{|T|} \),

(ii) moreover, if \( A : \text{Type}, a : A \vdash_{\text{id}} W : T \) then \( W \equiv_{\beta} \omega_{|T|} \).

**Proof.** We proceed by induction on the derivation of \( A : \text{Type}, a : A \vdash_{\text{id}} T : \text{Type}_0 \). We notice that the only two possible last used rules are IDENTITY and VARIABLE since \( \text{Type}_0 \) does not inhabit \( \text{Type}_0 \) it is not possible to invoke EXTENDED-PATH-INDUCTION nor CONVERSION.

(i) In the VARIABLE case, we necessarily have \( T \equiv A \) and therefore we can conclude by taking \( |T| = 0 \). In the IDENTITY case, \( T \) is of the shape \( M =_C N \) and by induction hypothesis \( M \equiv_{\beta} \omega_{|C|} \), \( N \equiv_{\beta} \omega_{|C|} \) and \( C \equiv_{\beta} \Omega_{|C|} \). So we conclude by taking \( |T| = |C| + 1 \).

(ii) Without loss of generality (MLTT is normalizing) we can assume that \( W \) is
Lemma 5.3

Proof.
We proceed by induction on size of derivations in

- **VARIABLE:** In this case, we necessarily have $W \equiv a$ and $T \equiv A$. So we have $|T| = 0$ and $W \equiv \omega_0$.

- **REFLEXIVITY:** In this case, $W$ and $T$ are respectively of the shape $1^C_M$ and $M \equiv C^M$. By induction hypothesis, we have $M \equiv \omega |C|$. Therefore $|T| = |C| + 1$ and $W \equiv \omega |C| + 1$.

- **EXTENDED-PATH-INDUCTION:** This is in fact an impossible case. We would have $W$ of the shape $J_{x:C,y:M=x,\Delta,p}(B,U,V,Z)$ and by induction hypothesis, we would have $M \equiv \omega |C| \equiv U$ and $V \equiv \omega |C| + 1$. Therefore $V$ is a reflexivity and so $W$ is not in normal form.

The previous lemma allow us to find a canonic instantiation of any contractible context given by:

$$\Gamma, x : A, y : M = C x \vdash (A, a)$$

The following lemma state that this instantiation is correct:

**Lemma 5.3**

$$\Gamma \text{ contr implies } A : \text{Type}_0, a : A \vdash \Gamma^+ : \Gamma \quad (1)$$

$$\Gamma \vdash T : \text{Type}_0 \text{ implies } T[\Gamma^+ / \Gamma] \equiv \Omega_{(T[\Gamma^+ / \Gamma])} \quad (2)$$

$$\Gamma \vdash T : \text{Type}_0 \text{ and } \Gamma \vdash M : T \text{ implies } M \equiv \omega |T[\Gamma^+ / \Gamma]| \quad (3)$$

**Proof.** We proceed by induction on size of derivations in MLID. We prove (1) by inspecting the last possible rule:

- **INITIAL:** $\Gamma$ is of the shape $A : \text{Type}_0, a : A$ and $\Gamma^+ = (A, a)$. By using two times **VARIABLE**, we check that: $(A : \text{Type}_0, a : A) \vdash (A, a) : (A : \text{Type}_0, a : A)$.

- **CONTRACTIBLE:** $\Gamma$ is of the shape $\Delta, x : B, p : M = B x$ with $\Delta \vdash B : \text{Type}_0$ and $\Gamma \vdash M : B$. By induction hypothesis, we have $A : \text{Type}_0, a : A \vdash \Delta^+ : \Delta$, in normal form. We proceed by a (nested) induction on the derivation $A : \text{Type}_0, a : A \vdash W : T$, we treat each possible case (IDENTITY and CONVERSION are obviously impossible since $T$ cannot be $\text{Type}_0$):

  - **VARIABLE:** In this case, we necessarily have $W \equiv a$ and $T \equiv A$. So we have $|T| = 0$ and $W \equiv \omega_0$.
  - **REFLEXIVITY:** In this case, $W$ and $T$ are respectively of the shape $1^C_M$ and $M \equiv C^M$. By induction hypothesis, we have $M \equiv \omega |C|$. Therefore $|T| = |C| + 1$ and $W \equiv \omega |C| + 1$.
  - **EXTENDED-PATH-INDUCTION:** This is in fact an impossible case. We would have $W$ of the shape $J_{x:C,y:M=x,\Delta,p}(B,U,V,Z)$ and by induction hypothesis, we would have $M \equiv \omega |C| \equiv U$ and $V \equiv \omega |C| + 1$. Therefore $V$ is a reflexivity and so $W$ is not in normal form.

Fig. 3. Examples of groupoid laws with aliases for canonical inhabitants.
Lemma 5.4 (All groupoid laws are inhabited) $\theta$ exists $\theta$ such that $T \vdash A \Delta \equiv (\Delta, x : B, p : M = B x)$.

Proof. The contractible context $A \theta$ obtain $\theta$ we construct $\theta$ after all the inductions, it remains to find an inhabitant of $\beta_T \equiv \Omega[T[\Gamma^+ / \Gamma]]$. By previous lemma.

To prove (2) and (3), we notice that within the derivation of $\Gamma \vdash_T T : \text{Type}_0$ there is a strictly smaller derivation of $\Gamma \text{contr}$ so we can use the induction hypothesis to obtain $A : \text{Type}_0, a : A \vdash_T \Gamma^+ : \Gamma$. Now by substitution, we derive that $A : \text{Type}_0, a : A \vdash_T M[\Gamma^+ / \Gamma] : T[\Gamma^+ / \Gamma]$. We conclude that $T[\Gamma^+ / \Gamma] \equiv \beta \Omega[T[\Gamma^+ / \Gamma]]$ and $M \equiv \beta \omega[\Gamma^+ / \Gamma]$ by previous lemma.

**Lemma 5.4 (All groupoid laws are inhabited)** If $\Gamma \vdash_T T : \text{Type}_0$, then there exists $\theta_T$ such that $\Gamma \vdash_T \theta_T : T$.

**Proof.** The contractible context $\Gamma$ is of the shape

$$A : \text{Type}_0, a : A, x_1 : C_1, y_1 : M_1 = x_1, \ldots, x_n : C_n, y_n : M_n = x_n$$

we construct $\theta_{\Gamma, c}$ by $n$ successive extended path-inductions (from left to right). After all the inductions, it remains to find an inhabitant of $T[\Gamma^+ / \Gamma]$ in the context $A : \text{Type}_0, a : A$. But thanks to the previous lemma, we know that $T[\Gamma^+ / \Gamma] \equiv \beta \Omega[T[\Gamma^+ / \Gamma]]$, therefore using CONVERSION we can use $\omega[\Gamma^+ / \Gamma]$. Spelled out, the term $\theta_{T, \Gamma}$ is :

$$\theta_{\Gamma, T} \equiv J_{x_1 : C_1, y_1 : M_1 = x_1, \ldots, x_n : C_n, y_n : M_n = x_n, T}(\ldots J_{x_n : C_n, y_n : M_n = x_n, T}[\Delta^+/\Delta](\omega[\Gamma^+ / \Gamma], x_n, y_n) \ldots, x_2, y_2), x_1, y_1)$$

where $\Delta$ is the context such that $\Gamma = \Delta, x_n : C_n, y_n : M_n = x_n$. 

As a corollary, we obtain the following theorem :

**Theorem 5.5 (Canonicity for groupoid laws in MLID)** If $\Gamma \vdash_T T : \text{Type}$ and if we have two terms $M$ and $N$ such that $\Gamma \vdash_T M : T$ and $\Gamma \vdash_T N : T$, then there is a proof $\pi$ such that $\Gamma \vdash \pi : M = N$.

**Proof.** Simply notice that $\Gamma \vdash M = N : \text{Type}_0$ and apply previous lemma.

The reader should remark here that the proof $\pi$ is also unique up to equality (by applying the theorem to $M = N$ !). We are now ready to show that previous theorem may be generalized to the whole system MLTT by using parametricity theory.

**Theorem 5.6 (Canonicity of inhabitants of groupoid laws in MLTT)** If $\Gamma \vdash_T T : \text{Type}$, $\vdash_T M : \forall \Gamma T$ and $\vdash_T N : \forall \Gamma T$, then there is a proof $\pi$ such that $\Gamma \vdash \pi : (M \gamma) = (N \gamma)$.

**Proof.** By successive extended path-inductions (from left to right), we can derive $\gamma' : \Gamma \vdash (M \gamma') = (N \gamma')$ from a derivation of $A : \text{Type}_0, a : A \vdash \gamma'$, and $\gamma' = (N \gamma')$. We notice that the type of $(M \Gamma^+)$ is $T[\Gamma^+ / \Gamma]$ which is typable in MLID; by substitution we have $A : \text{Type}_0, a : A \vdash_T T[\Gamma^+ / \Gamma] : \text{Type}_0$. Therefore Lemma 5.3
gives us that $T[\Gamma^{+}/\Gamma] \equiv_{\beta} \Omega_{n}$ for some $n \in \mathbb{N}$. Using conversion, we have $A : \text{Type}_{0}, a : A \vdash (M \Gamma^{+}) : \Omega_{n}$. Therefore $(M \Gamma^{+})$ is an inhabitant of a parametric loop space; therefore we can invoke Theorem 4.3 to obtain of proof that $(M \Gamma^{+}) = \omega_{n}$. Similarly we have a proof of $(N \Gamma^{+}) = \omega_{n}$ and by concatenating them we obtain a proof of $(M \gamma) = (N \gamma)$.

Finally, using the same arguments as at the end of Section 4 we can prove that $\pi$ is unique up to propositional equality (and so on).

6 Discussions

6.1 The definition of groupoid laws

Our definition of MLID is inspired by an unpublished note written by Brunerie [3]. Our syntax for contractible contexts is a bit more general: in Brunerie’s definition the starting point of paths occurring at odd positions are always variable (ie. $M_{k}$ is always a variable) and there is no computation rules in his syntax. Brunerie defines $\omega$-groupoids as models of its syntax, since the absence of computation rules makes his framework more free about how coherence issues are dealt with. However, the goal of our syntax is not give the general syntax for weak $\omega$-groupoids but rather to study only the groupoid structure in the particular case of type theory where computation rules are the natural way to deal with coherence. Nevertheless, it would be an interesting future work to make a precise comparison between other definitions of $\omega$-groupoids and models of MLID.

The semantical nature of Garner and van den Berg [9] makes it quite difficult to relate to our work, however we believe that Lumsdaine’s construction [5] of a contractible globular operad may be described in our framework.

6.2 The $n$-ary case

Throughout this article, we only use the unary case of parametricity theory, but it could be easily generalized to the binary case by transporting along two paths: $[x = y]_{2} \equiv \lambda(p : x = y)(q : x' = y'). p \ast (q \ast (x R)) = y R$. This translation is well-typed under the binary translation $[\alpha : \text{Type}_{i}, x : \alpha, y : \alpha]_{2}$ given by

$$\alpha \alpha' : \text{Type}_{i}, \alpha R : \alpha \rightarrow \alpha' \rightarrow \text{Type}_{i}, x : \alpha, x' : \alpha', x R : \alpha R x x', y : \alpha, y' : \alpha', y R : \alpha R y y'$$

The translation $[1^{n}_{2}]_{2} : 1^{n}_{2}(1^{n}_{2}(x R)) = x R$ of $1^{n}_{2}$ is given by $[1^{n}_{2}]_{2} \equiv 1^{n}_{2} x x$ which is defined under the translated context $[\alpha : \text{Type}_{i}, x : \alpha]_{2}$. Similarly, the translation of path-induction is obtained by nesting $2+1$ path-induction:

$$[J(B, U, V)]_{2} \equiv J(J([B]_{2}, U, V), U', V'), [U]_{2}, [V]_{2}$$

It is a routine check to generalize the abstraction theorem of Section 3, in order to a have a binary version of parametricity. Likewise, the $n$-ary case, is obtained by transporting along $n$ path and the translation of path-induction is obtained by nesting $n + 1$ path-inductions.
6.3 Encoding identity types with inductive families.

In dependent type systems that support inductive families, it is possible to encode identity types by an inductive predicate [4]. For instance, in the coq proof assistant:

\[
\text{Inductive paths } (A : \text{Type}) (a : A) : A \to \text{Type} := \text{idpath} : \text{paths } A \ a \ a.
\]

As explained in [2] (Section 5.4), the parametricity translation extends well to inductive families. The idea is to translate an inductive type \(I\) by a new inductive \(I_R\) whose constructors are the translation of constructors of \(I\). Likewise, the elimination scheme for \(I_R\) is the translation of the one of \(I\).

\[
\text{Inductive paths}_R (A : \text{Type}) (A_R : A \to \text{Type}) (a : A) (a_R : A_R a) : \forall x, A_R x \to \text{paths } A \ a x \to \text{Type} := \text{idpath}_R : \text{paths}_R A A_R a a_R a a_R (\text{idpath } A \ a).
\]

There is an equivalence of types (in the homotopy theory sense, see [8]) between this inductive type and our translation of identity. It is obtained by using the induction principle associated with \(\text{paths}_R\) in one direction; and by nested path-inductions on \(p\) and on the proof of transport along \(p\) in the other direction. This indicates that they are morally the same; it may convince the reader that the results of the last two sections could have been carried out using the encoding instead of the primitive notion of identity types.

6.4 Dealing with axioms

While formalizing proofs that need axioms which are independent, it is a common practice to simply add them in the context. Then, if one want to use the parametricity translation, he also needs to provide a witness of the parametricity of the axiom. Therefore axioms that can prove their own parametricity are well-behaved with respect to the translation. More formally, we say that a closed type \(P : \text{Type}\) is provably parametric if the type \(\forall h : P, h \in \llbracket P \rrbracket\) is inhabited. We will now give two examples of axioms using identity types which are provably parametric.

- uniqueness of identity proofs (UIP) : Let \(\text{uip}\) be the following type \(\text{uip} \equiv \forall X : \text{Type}, x y : X, p q : x = y.p = q\). We want to find an inhabitant of \(\forall f : \text{uip}, f \in \llbracket \text{uip} \rrbracket\). The statement \(f \in \llbracket \text{uip} \rrbracket\) unfolds into \(\forall [X : \text{Type}, x y : X, p q : x = y], (f A x y p q)_(p_R) = q_R\). The conclusion is an equality between paths, so it is provable using \(f\).

- function extensionality : Let \(\text{funext}\) be the following type \(\text{funext} \equiv \forall A : \text{Type}, B : A \to \text{Type}, f g : \forall x : A.Bx. (\forall x : A.f x = g x) \to f = g\). In his original development, Voedvoesky showed that \(\text{funext}\) is logically equivalent (there is a function in both directions) to the so-called weak extensionality (see [8]). It is defined by \(\text{weakext} \equiv \forall A : \text{Type}, P : A \to \text{Type}. (\forall x : A.\text{Contr} P x) \to \text{Contr} (\forall x : A.P x)\) where \(\text{Contr} A \equiv \exists x : A.\forall y : A.x = y\) is the predicate for contractible types; ie. types which are equivalent to the singleton type. We now sketch the proof that \(\text{weakext}\) is provably parametric (and thus so is \(\text{funext}\)). The main idea is to notice that \(M \in \llbracket \text{Contr} A \rrbracket\) is logically equivalent to \(\text{Contr} ([A] M_1)\)
where $M_1$ is the first projection of $M$. The unfolding of $k \in \mathbb{weakext}$ is:

$$
\forall A : Type, A_R : A \to Type, P : A \to Type, P_R : (\forall x:A, x_R : A_R x.P x \to Type), \phi : (\forall x:A.\text{Contr}(P x)), \phi_R : (\forall x:A, x_R : A_R x.(\phi x) \in \llbracket \text{Contr} P x \rrbracket).
$$

$$(kA P \phi) \in \llbracket \text{Contr}(\forall x:A.P x) \rrbracket$$

Therefore, using the logical equivalence in one direction we can deduce the conclusion from $\text{Contr}(kA P \phi)_1 \in \llbracket \forall x:A.P x \rrbracket)$. Then using $k : \mathbb{weakext}$ two times, it is enough to prove that $\forall x : A, x_R : A_R x.\text{Contr}((kA P \phi)_1 x) \in \llbracket P x \rrbracket$ (1). Notice that $(kA P \phi)_1 x = A (\phi x)_1$ because $A$ is contractible. So we can transport along this path in (1) to obtain $\forall x : A, x_R : A_R x.\text{Contr}((\phi x)_1 \in \llbracket P x \rrbracket)$ (2). Now, using the other direction of logical implication, (2) is implied by $\forall x : A, x_R : A_R x.(\phi x) \in \llbracket \text{Contr} P x \rrbracket)$ which is exactly the type of $\phi_R$.

7 Conclusion

This work shows that parametricity theory may be used to deduce properties about the algebraic structure of identity types. It allows to give formal arguments to prove canonicity results about definitions in a proof-relevant setting.

The most important question that remains open is whether or not we can extend the translation and the uniqueness property of groupoid laws to deal with Voevodsky’s univalence axiom. One way to solve this question would be to prove that univalence axiom is provably parametric which would yield to a positive answer to the question of the compatibility of parametricity theory and univalence. Regardless of the answer to this problem, solving it would give a better understanding of polymorphic type quantifications in univalent universes.

References

Abstract

We define the class of relational graph models and study the induced order- and equational- theories. Using the Taylor expansion, we show that all $\lambda$-terms with the same Böhm tree are equated in any relational graph model. If the model is moreover extensional and satisfies a technical condition, then its order-theory coincides with Morris's observational pre-order. Finally, we introduce an extensional version of the Taylor expansion, then prove that two $\lambda$-terms have the same extensional Taylor expansion exactly when they are equivalent in Morris’s sense.

Keywords: lambda calculus, linear logic, differential nets, extensional Böhm trees, Taylor expansion.

Introduction

An important problem in the theory of programming languages is to determine when two programs are equivalent. For $\lambda$-calculus, it has become standard to regard two programs $M$ and $N$ as equivalent when they are contextually equivalent with respect to some fixed set $O$ of observables. This means that we can plug either $M$ or $N$ into any context $C(\_)$, i.e. any program with a hole, without noticing any difference in the global behaviour: $C(M)$ reduces to an observable in $O$ exactly when $C(N)$ does.

Two notable examples are $\equiv^{hnf}$ and Morris’s equivalence $\equiv^{nf}$ [17] obtained by taking as observables the head normal forms and the $\beta$-normal forms, respectively. Working with these definitions is difficult because of the quantification over all possible contexts. However, researchers have found alternative characterisations of these program equivalences based on syntactic trees or denotational models.

For instance, two programs are equivalent with respect to $\equiv^{hnf}$ whenever they have the same Nakajima tree [18] or, equivalently, when their interpretations coin-
cidc in Scott’s model $D_{∞}$ [21]. Similarly, $≡^nf$ is captured by extensional Böhm trees [14] and Coppo, Dezani and Zacchi’s filter model $D_{cdz}$ [6].

The idea behind Böhm trees, and their extensional versions, is to extract the computational content of a program by representing its output as a possibly infinite tree — the continuity of this representation allows to infer properties of the whole tree by studying its finite approximants. For this reason Böhm-like trees and continuous models relied to them via approximation theorems constituted for over forty years the main tools to reason about the behaviour of a program. A limitation of these methods is that they abstract away from the execution process and overlook quantitative aspects such as the time, space, or energy consumed by a computation.

The present paper fits in a wider research programme whose aim is to rebuild the traditional theory of program approximations, by replacing it with a mathematical model of resource consumption. The starting point is [9], where Ehrhard and Regnier propose to analyse the behaviour of a program via its Taylor expansion, which is a generally infinite series of “resource approximants”. Such approximants are terms of a resource calculus corresponding to a finitary fragment of the differential $λ$-calculus [7]. Each resource approximant $t$ of a $λ$-term $M$ captures a particular choice of the number of times $M$ must call its sub-routines during its execution.

Both the differential $λ$-calculus and the Taylor expansion can be naturally interpreted in the relational semantics of linear logic [16]. The first author et al. built a relational model $D_ω$ living in such a semantics [5] and proved, using standard techniques, that the induced equality is exactly $≡^hnf$ [15], just like for Scott’s model $D_{∞}$ [12]. In this paper we provide syntactical and denotational methods based on Taylor expansion that allow to characterise Morris’s equivalence $≡^nf$.

First, we introduce the class of relational graph models (rgms) of $λ$-calculus, which are the relational analogous of graph models [3], and describe them as non-idempotent intersection type systems [19]. This class is general enough to encompass all relational models individually introduced in the literature [5,13], including $D_ω$ (while Scott’s $D_{∞}$ cannot be a graph model since it is extensional). We then show that: (i) all rgms satisfy an approximation theorem for resource approximants (Theorem 3.10); (ii) in any rgm preserving the polarities of its “empty type” $ω$, $β$-normalisable $λ$-terms can be easily characterized (Lemma 4.3). As a consequence, we get that all extensional rgms preserving $ω$-polarities induce as order-theory Morris’s observational pre-order, and hence $≡^nf$ as equality (Corollary 4.6). As an instance, we provide the rgm $D_*$ generated by $* \to * \simeq *$ where $*$ is the only atom. It should be compared with the aforementioned filter model $D_{cdz}$, which has the same theory but is more complicated since it has two non-trivially ordered atoms $φ_{↑} ≤ φ_*$ and is generated by two equations $φ_{↑} \simeq φ_* \to φ_{↑}$ and $φ_* \simeq φ_{↑} \to φ_*$.

Finally, we provide a notion of extensional Taylor expansion characterising, like extensional Böhm trees, Morris’s equivalence while keeping the quantitative information. Intuitively, the extensional Taylor expansion of a $λ$-term is the $η$-normal form of its resource approximants. The definition is tricky because the $η$-reduction is meaningless on a single resource approximant — one should look at the whole series of approximants to decide whether an element should reduce or not. Our solution is
to define a labeling as a global operation on the series of approximants, and then a local \( \eta \)-reduction on labeled terms. Two programs are then \( \equiv_{\text{nf}} \)-equivalent exactly when they have the same extensional Taylor expansion (Theorem 5.17). We leave for future works a characterisation of Morris’s preorder based on Taylor expansion.

**Basic notations and conventions.** We let \( \mathbb{N} \) denote the set of natural numbers. Given a set \( A \), \( \mathcal{P}(A) \) (resp. \( \mathcal{P}_f(A) \)) is the set of all (resp. finite) subsets of \( A \) and \( \mathcal{M}_f(A) \) is the set of all finite multisets over \( A \). Finite multisets are represented as unordered lists \( m = [\alpha_1, \ldots, \alpha_n] \) with repetitions, \([\,]\) being the empty multiset.

Given a reduction \( \rightarrow_r \) we write \( \rightarrow^{=}_r \) for its transitive and reflexive (and symmetric) closure. A term \( t \) has an \( r \)-normal form \( \text{nf}_r(t) \), if \( t \rightarrow^{=}_r \text{nf}_r(t) \).

**N.B.** Unless otherwise stated, throughout the paper we suppose that all operators \( F : A \rightarrow B \) are extended to \( \mathcal{P}(A) \) in the natural way: \( F(a) = \{ F(\alpha) \mid \alpha \in a \} \).

# 1 Lambda Calculus and Böhm Trees

We will generally use the notation of Barendregt’s classic work [2] for \( \lambda \)-calculus. Let us fix an infinite set \( \text{Var} \) of variables. The set \( \Lambda \) of \( \lambda \)-terms is defined by:

\[
\Lambda : \quad M, N, P ::= x \mid \lambda x.M \mid MN \quad \text{for all } x \in \text{Var}.
\]

The set \( \text{fv}(M) \) of free variables of \( M \) and the \( \alpha \)-conversion are defined as usual, see [2, Ch. 1][2]. A \( \lambda \)-term \( M \) is closed if \( \text{fv}(M) = \emptyset \). We denote by \( \Lambda^\circ \) the set of closed \( \lambda \)-terms. From now on, \( \lambda \)-terms will be considered up to \( \alpha \)-conversion.

Given two \( \lambda \)-terms \( M, N \) we denote by \( M\{N/x\} \) the capture-free substitution of \( N \) for all free occurrences of \( x \) in \( M \). The \( \beta \)- and \( \eta \)-reductions are given for granted.

Concerning specific \( \lambda \)-terms, we fix the identity \( I = \lambda x.x \), its \( \eta \)-expansion \( I = \lambda x y.x y \), the paradigmatic looping term \( \Omega = \Delta \Delta \) where \( \Delta = \lambda x.x x \), Turing’s fixpoint combinator \( \Theta = \lambda f.\Theta f\Theta f \) where \( \Theta f = \lambda x.f(f(x)) \) and \( J = \Theta(\lambda z x y.z y) \) a term reducing to an infinite \( \eta \)-expansion of \( I \).

A \( \lambda \)-term \( M \) is called solvable if it has a head normal form (hnf, for short), that is if \( M \rightarrow^\beta \lambda x_1 \ldots x_n.y N_1 \cdots N_k \) (for \( n, k \geq 0 \)); otherwise \( M \) is called unsolvable.

Given a context \( C(-) \), i.e. a \( \lambda \)-term with a hole denoted by \( (\_\_\_\_\_) \), we write \( C(M) \) for the \( \lambda \)-term obtained from \( C \) by substituting \( M \) for the hole possibly with capture of free variables in \( M \). Given \( \mathcal{O} \subseteq \Lambda \), the \( \mathcal{O} \)-observational pre-order is defined by:

\[
M \sqsubset^\mathcal{O} N \iff \forall C(-). C(M) \rightarrow^\beta M' \in \mathcal{O} \text{ entails } C(N) \rightarrow^\beta N' \in \mathcal{O}.
\]

The induced equivalence \( M \equiv^\mathcal{O} N \) is defined as \( M \sqsubset^\mathcal{O} N \) and \( N \sqsubset^\mathcal{O} M \). To obtain Morris’s pre-order \( \equiv_{\text{nf}} \) and equivalence \( \equiv_{\text{nf}} \) just take as \( \mathcal{O} \) the set of \( \beta \)-nfs [17].

The Böhm tree \( \text{BT}(M) \) of a \( \lambda \)-term \( M \) is defined coinductively: if \( M \) is unsolvable then \( \text{BT}(M) = \bot \); if \( M \) is solvable, then \( M \rightarrow^\beta \lambda x_1 \ldots x_n.y N_1 \cdots N_k \) and

\[
\text{BT}(M) = \begin{cases} 
\lambda x_1 \ldots x_n.y & \text{if } M \text{ is solvable,} \\
\text{BT}(N_1) & \cdots \\
\text{BT}(N_k)
\end{cases}
\]

249
Such a definition is sound in the sense that $M =_\beta N$ entails $BT(M) = BT(N)$.
Examples of Böhm trees are: $BT(I) = 1$, $BT(1) = 1$, $BT(\Delta) = \Delta$, $BT(\Omega) = \perp$,
$$BT(\lambda x.y\Omega) = \lambda x.y \quad BT(J) = \lambda xz_0.x \quad BT(\Theta) = \lambda f.f$$

Given two Böhm trees $T,T'$ we set $T \leq_{\perp} T'$ if and only if $T$ results from $T'$ by
replacing some subtrees with $\perp$. The set $\mathcal{N}$ of finite approximants is the set of
$\lambda$-terms possibly containing $\perp$ inductively defined as follows: $\perp \in \mathcal{N}$; if $a_i \in \mathcal{N}$ for
$i = 1, \ldots, n$ then $\lambda x.ya_1 \cdots a_n \in \mathcal{N}$. Hereafter we will confuse finite Böhm trees
with normal approximants. Notice that the set of all finite approximants of a Böhm
tree $T$, given by $T^* = \{a \in \mathcal{N} \mid a \leq_{\perp} T\}$, is an ideal with respect to $\leq_{\perp}$ [1, §2.3].

A $\lambda$-theory is any congruence on $\Lambda$ containing $=_{\beta}$. A $\lambda$-theory is: extensional if
it contains $=_{\eta}$; sensible if it equates all unsolvables. We denote by: $\lambda\beta\eta$ the least
extensional $\lambda$-theory; $B$ the $\lambda$-theory equating all $\lambda$-terms having the same Böhm
tree; $B\eta$ the least $\lambda$-theory containing $B$ and $\lambda\beta\eta$; $H^+$ (resp. $H^*$) the $\lambda$-theory
characterizing $\equiv^{nf}$ (resp. $\equiv^{finf}$). From [2, Thm. 17.4.16] we get $B \subseteq B\eta \subseteq H^+ \subseteq H^*$.

## 2 Resource Calculus and Taylor Expansion

We briefly recall Ehrhard’s resource calculus [8], using the syntax proposed by Tran-
quilli in [22]. We are considering here the promotion-free fragment of [22].

**Syntax.** The set $\Lambda^r$ of resource terms and the set $\Lambda^b$ of bags are defined by:

$$\Lambda^r : \quad s, t ::= x \mid \lambda x.t \mid tb$$

$$\Lambda^b : \quad b ::= [s_1, \ldots, s_n] \text{ where } n \geq 0. \quad (1)$$

Resource terms are in functional position, while bags are in argument position
and represent unordered lists of resource terms. Intuitively, in a term of shape
$t([s_1, \ldots, s_n])$ each $s_i$ is a linear resource, that is $t$ cannot duplicate nor erase it.

We will deal with bags as if they were multisets presented in multiplicative
notation: $1$ is the empty bag and $b_1 \cdot b_2$ is the multiset union of $b_1$ and $b_2$.

We use the power notation $[s^k]$ for the bag $[s, \ldots, s]$ containing $k$ copies of $s$.

The $\alpha$-equivalence and the set $fv(t)$ of free variables of $t$ are defined as for the
ordinary $\lambda$-calculus. Resource terms and bags are considered up to $\alpha$-equivalence.

As a syntactic sugar, we extend all the constructors of the grammar (1) as
pointwise operations on (possibly infinite) sets of resource terms or bags. That is,
given $T \subseteq \Lambda^r$ and $B, B' \subseteq \Lambda^b$ we use the following notations:
$$\lambda x.T = \{\lambda x.t \mid t \in T\}, \quad TB = \{tb \mid t \in T, b \in B\}, \quad [T] = \{[t] \mid t \in T\} \quad \text{and} \quad B \cdot B' = \{b \cdot b' \mid b \in B, b' \in B'\}.$$

Observe that, in the particular case of empty set, we get $\lambda x.\emptyset = \emptyset$, $t\emptyset = \emptyset$, $\emptyset b = \emptyset$, $[\emptyset] = \emptyset$ and $\emptyset \cdot b = \emptyset$. Hence, $\emptyset$ annihilates any resource term or bag.

This kind of meta-syntactic notation is discussed thoroughly in [9].

**Reductions.** Given a relation $\rightarrow =_{\perp} \subseteq \Lambda^r \times \mathcal{P}(\Lambda^r)$ its context closure is the least
relation in $\mathcal{P}(\Lambda^r) \times \mathcal{P}(\Lambda^r)$ such that, when $t \rightarrow =_{\perp} T$, we have:

$$t \rightarrow =_{\perp} T$$

250
\[ \lambda x. t \rightarrow_\pi \lambda x. T, \quad t b \rightarrow_\pi T b, \quad s([t] \cdot b) \rightarrow_\pi s([T] \cdot b), \quad \{t\} \cup S \rightarrow_\pi T \cup S. \]

We say that \( t \in \Lambda^r \) is in \( \pi \)-normal form if there is no \( T \) such that \( t \rightarrow_\pi T \). When \( \rightarrow_\pi \) is confluent, \( \text{nfr}(t) \in P_f(\Lambda^r) \) denotes the unique \( \pi \)-normal form of \( t \), if it exists.

The degree of \( x \) in \( t \), written \( \text{deg}_x(t) \), is the number of free occurrences of \( x \) in \( t \). A \( \beta \)-redex is a resource term of the shape \( (\lambda x. t)[s_1, \ldots, s_k] \) and its contractum is a finite set of resource terms: when \( \text{deg}_x(t) = k \), it is the set of all possible resource terms obtained by linearly replacing each free occurrence of \( x \) in \( t \) by exactly one of the \( s_i \)'s; otherwise, when \( \text{deg}_x(t) \neq k \), it is just \( \emptyset \).

Formally, we define \( \rightarrow_\beta \) as the context closure of:

\[
(\lambda x. t)[s_1, \ldots, s_k] \rightarrow_\beta \begin{cases} 
\bigcup_{p \in S_k} t\{s_p(1)/x_1, \ldots, s_p(k)/x_k\} & \text{if } \text{deg}_x(t) = k, \\
\emptyset & \text{otherwise.}
\end{cases}
\]

where \( S_k \) is the group of permutations of \( \{1, \ldots, k\} \) and \( x_1, \ldots, x_n \) is an arbitrary enumeration of the free occurrences of \( x \) in \( t \). Note that \( \beta \)-reduction is strongly normalizing (SN, for short) on \( P_f(\Lambda^r) \), since whenever \( t \rightarrow_\beta T \) the size of \( t \) is strictly bigger than the size of each resource term in \( T \). Moreover, \( \beta \)-reduction is weakly confluent, and therefore confluent by Newman’s lemma.

**Theorem 2.1** The \( \beta \)-reduction is strongly normalizing and confluent on \( P_f(\Lambda^r) \).

In the resource calculus there is no sensible notion of \( \eta \)-reduction on \( P_f(\Lambda^r) \).

**Taylor expansion.** The Taylor expansion of a \( \lambda \)-term, as defined in [7,9], is a translation developing every \( \lambda \)-calculus application as an infinite series of resource applications with rational coefficients. For our purpose it is enough to consider a simplified version \( \mathcal{T}(\_): \Lambda \rightarrow P(\Lambda^r) \) corresponding to the support\(^4\) of the actual Taylor expansion; that is, we consider possibly infinite sets of resource \( \lambda \)-terms.

**Definition 2.2** The Taylor expansion \( \mathcal{T}(M) \subseteq \Lambda^r \) of a \( \lambda \)-term \( M \) is defined by:

\[
\mathcal{T}(x) = x, \quad \mathcal{T}(\lambda x. M) = \lambda x. \mathcal{T}(M), \quad \mathcal{T}(MN) = \mathcal{T}(M) \mathcal{T}(N).
\]

The Taylor expansion is extended to finite approximants in \( \mathcal{N} \) by setting \( \mathcal{T}(\bot) = \emptyset \), and to Böhm trees \( T \) by setting \( \mathcal{T}(T) = \bigcup\{\mathcal{T}(a) \mid a \in T^*\} \).

Some examples of Taylor expansions of ordinary \( \lambda \)-terms are:

\[
\mathcal{T}(I) = \{I\}, \quad \mathcal{T}(\Delta) = \{\lambda x.x^n \mid n \geq 0\}, \quad \mathcal{T}(\lambda y. x y y) = \{\lambda y.x[y^n][y^k] \mid n, k \geq 0\},
\]

\(^4\) I.e., the set of those resource terms appearing in the series with a non-zero coefficient.

251
Manzonetto & Ruoppolo

\[ \mathcal{T}(\Omega) = \{ (\lambda x.x[x^0]) [\lambda x.x[x^1], \ldots, \lambda x.x[x^n]] : k, n_0, \ldots, n_k \geq 0 \}, \]
\[ \mathcal{T}(\Theta) = \{ \lambda f. (\lambda x.f[x[x^1], \ldots, x[x^n]]) [\lambda x.f[x[x^1], \ldots, x[x^n, k_1]]], \ldots, \lambda x.f[x[x^n, k_1]], \ldots \}, \]
\[ \mathcal{T}(J) = \{ t[\lambda zxy.x[z[y^n, k_1]], \ldots, z[y^n, k_1]], \ldots], \]
\[ \lambda zxy.x[z[y^n, k_1]], \ldots, z[y^n, k_1]] : t \in \mathcal{T}(\Theta), h, k_i, n_i, j \geq 0 \}. \]

From the examples above it is clear that if a \( \lambda \)-term \( M \) has a \( \beta \)-redex, then there are resource terms \( t \in \mathcal{T}(M) \) having \( \beta \)-redexes too. However, by Theorem 2.1, each \( t \) has a unique \( \beta \)-nf and we can always compute \( \text{nf}_\beta(\mathcal{T}(M)) = \bigcup \{ \text{nf}_\beta(t) : t \in \mathcal{T}(M) \} \). For instance: \( \mathcal{T}(I), \mathcal{T}(\Delta) \) and \( \mathcal{T}(\lambda y.xyy) \) are already \( \beta \)-normal, while \( \text{nf}_\beta(\mathcal{T}(\Omega)) = \emptyset \).

**Lemma 2.3** Let \( a \in N \) and \( M \in \Lambda \), then \( \mathcal{T}(a) \subseteq \mathcal{T}(\text{BT}(M)) \) entails \( a \in \text{BT}(M)^* \).

The following results proved in [8] show the strong relationship between the Böhm tree of a \( \lambda \)-term, and its Taylor expansion.

**Theorem 2.4** For every \( \lambda \)-term \( M \), \( \text{nf}_\beta(\mathcal{T}(M)) = \mathcal{T}(\text{BT}(M)) \).

**Corollary 2.5** For all \( M, N \in \Lambda \), \( \text{BT}(M) = \text{BT}(N) \) iff \( \text{nf}_\beta(\mathcal{T}(M)) = \text{nf}_\beta(\mathcal{T}(N)) \).

Using Theorem 2.4, we can easily calculate further examples:
\[ \text{nf}_\beta(\mathcal{T}(\Theta)) = \{ \lambda f. f, 1, \lambda f. f[(f1)^n], \lambda f. f[f[(f1)^n]], \ldots, f[(f1)^n]], \ldots \}, \]
\[ \text{nf}_\beta(\mathcal{T}(J)) = \{ \lambda xz_0.x1, \lambda xz_0.x[(\lambda z_1.z0)^n]], \ldots \}. \]

## 3 Relational Graph Models and Intersection Types

In this section we introduce the class of relational graph models (rgm, for short); some examples of such models were individually studied in [13].

### 3.1 Relational Graph Models

We call rgmms *relational* because they are (linear) reflexive objects in the ccc MRRel [5], the Kleisli category of Rel with respect to the comonad \( \mathcal{M}_t(\cdot) \). In MRRel the objects are all the sets, a morphism \( f \in \text{MRRel}(A, B) \) is any relation between \( \mathcal{M}_t(A) \) and \( B \), and the exponential object \( A \Rightarrow B \) is given by \( \mathcal{M}_t(A) \times B \). Any function \( f : A \to B \) can be sent to \( f^\dagger \in \text{MRRel}(A, B) \) by setting \( f^\dagger = \{ ([a], f(a)) : a \in A \} \).

**Definition 3.1** A relational graph model \( D = (D, i) \) is given by an infinite set \( D \) and a total injection \( i : \mathcal{M}_t(D) \times D \to D \). \( D \) is *extensional* when \( i \) is bijective.

Every rgm \( D = (D, i) \) induces a reflexive object \( (D, i^\dagger, (i^{-1})^\dagger) \), i.e. \( D \Rightarrow D \subset D \) since \( i^\dagger: (i^{-1})^\dagger = \text{id}_D \). When \( D \) is moreover extensional we also have \( (i^{-1})^\dagger ; i^\dagger = \text{id}_D \). These reflexive objects are all *linear* in the sense of [16] and live in a differential ccc; they are therefore sound models of the resource calculus as well (Theorem 3.8).

Rgms, just like the regular ones [3], can be built by performing the free completion of a partial pair. A partial pair \( A \) is a pair \((A, j)\) where \( A \) is a non-empty set of elements (called *atoms*) and \( j : \mathcal{M}_t(A) \times A \to A \) is a partial injection. We say
that $A$ is extensional when $j$ is a bijection between $\text{dom}(j)$ and $A$. Wlog., we will only consider partial pairs $A$ whose underlying set $A$ does not contain any pair.

**Definition 3.2** The completion $\overline{A}$ of a partial pair $A$ is the pair $(\overline{A}, \overline{j})$ defined as: $\overline{A} = \bigcup_{n \in \mathbb{N}} A_n$, where $A_0 = A$ and $A_{n+1} = ((\mathcal{M}_t(A_n) \times A_n) - \text{dom}(j)) \cup A$; the function $\overline{j}$ is given by $\overline{j}(a, \alpha) = j(a, \alpha)$ if $(a, \alpha) \in \text{dom}(j)$, $\overline{j}(a, \alpha) = (a, \alpha)$ otherwise.

Note that, for every $\text{rgm } D$ we have $\overline{D} = D$ (up to isomorphism).

**Proposition 3.3** If $A$ is a partial pair, then $\overline{A}$ is an $\text{rgm}$. When $A$ is extensional, also $\overline{A}$ is extensional.

**Proof** The proof of the fact that $\overline{A}$ is an $\text{rgm}$ is analogous to the one for regular graph models [3]. It is easy to check that when $j$ is bijective, also $\overline{j}$ is. \qed

**Example 3.4** We define the relational analogues of:

- Engeler's model [10]: $E = (\mathbb{N}, \emptyset)$, first defined in [13],
- Scott's model [21]: $D_\omega = (\{\varepsilon\}, \{([\varepsilon], \varepsilon) \mapsto \varepsilon\})$, first defined (up to iso) in [5],
- Coppo, Dezani and Zacchi's model [6]: $D_\ast = (\{\ast\}, \{([\ast], \ast) \mapsto \ast\})$.

Notice that $D_\omega$ and $D_\ast$ are extensional, while $E$ is not.

### 3.2 Non-Idempotent Intersection Type Systems

As discussed thoroughly in [19], the choice of presenting a relational model as a reflexive object or as a non-idempotent intersection type system is more a matter of taste rather than a technical decision. Here we provide the latter presentation.

Let $A$ be a partial pair and $D$ be its completion. The set $T_D$ of types and the set $I_D$ of non-idempotent intersections are defined by mutual induction (for $\alpha \in A$):

$$T_D : \quad \sigma, \tau ::= \alpha \mid \mu \rightarrow \sigma \quad I_D : \quad \mu, \nu ::= \omega \mid \sigma \mid \sigma \land \mu$$

Note that types are (unary) intersections while the converse does not hold; indeed intersections may only appear at the left-hand side of an arrow. Thus $\omega$ is not a type, it denotes the empty intersection and is therefore its neutral element ($\mu \land \omega = \mu$). Accordingly, we write $\land_{i=1}^n \sigma_n$ for $\sigma_1 \land \cdots \land \sigma_n$ when $n \geq 1$, and for $\omega$ when $n = 0$.

Types will be considered up to associativity and commutativity of $\land$ and neutrality of $\omega$, while we assume that the intersection is not idempotent, that is $\sigma \land \sigma \neq \sigma$.

Every $\sigma \in T_D$ ($\mu \in I_D$) corresponds to an element $\sigma^\bullet$ of $D$ ($\mu^\bullet$ of $\mathcal{M}_t(D)$) defined as $\alpha^\bullet = \alpha$, $(\mu \rightarrow \tau)^\bullet = \iota(\mu^\bullet, \tau^\bullet)$ and $(\sigma_1 \land \cdots \land \sigma_n)^\bullet = [\sigma_1^\bullet, \ldots, \sigma_n^\bullet]$. Hence, the model $D$ induces a congruence on the intersection types: $\sigma \simeq_D \tau$ if and only if $\sigma^\bullet = \tau^\bullet$.

An environment is a map $\Gamma : \text{Var} \rightarrow I_D$ such that $\text{dom}(\Gamma) = \{x \mid \Gamma(x) \neq \omega\}$ is finite. We write $x_1 : \mu_1, \ldots, x_n : \mu_n$ for the environment $\Gamma$ such that $\Gamma(x_i) = \mu_i$ and $\Gamma(y) = \omega$ for all $y \notin \vec{x}$. The environment mapping all variables to $\omega$ is denoted by $\emptyset$, or just omitted as in Example 3.6. The intersection $\Gamma_1 \land \Gamma_2$ and the equivalence $\Gamma_1 \simeq_D \Gamma_2$ of two environments are defined pointwise; note that $\Gamma \land \emptyset = \Gamma$.

**Definition 3.5** The interpretation of $M \in \Lambda$ (or $M \in \mathcal{N}$) in $D$ is defined as:
The intersection type systems for Λ, Figure 1:

\[
\begin{array}{lll}
x : \sigma \vdash \Gamma, x : \mu \vdash M : \sigma & \text{var} & \Gamma \vdash \lambda x. M : \mu \rightarrow \sigma \\
\Gamma \vdash \lambda x. M : \mu \rightarrow \sigma & \text{lam} & \Gamma \vdash \lambda x. M : \sigma \\
\end{array}
\]

\[
\begin{array}{lll}
\Gamma_0 \vdash M : \bigwedge_{i=1}^n \sigma_i \rightarrow \tau & \Gamma_i \vdash N : \sigma_i & \text{for } i = 1, \ldots, n \\
\Gamma_0 \land (\bigwedge_{i=1}^n \Gamma_i) \vdash MN : \tau & \text{app} \\
\end{array}
\]

(a) Non-idempotent intersection type system for Λ and \( \mathcal{N} \).

\[
\begin{array}{lll}
\Gamma_0 \vdash t : \bigwedge_{i=1}^n \sigma_i \rightarrow \tau & \Gamma_i \vdash s_i : \sigma_i & \text{for } i = 1, \ldots, n \\
\Gamma_0 \land (\bigwedge_{i=1}^n \Gamma_i) \vdash [s_1, \ldots, s_n] : \tau & \text{app'} \\
\end{array}
\]

(b) Non-idempotent intersection type system for \( \Lambda^r \).

Figure 1: The intersection type systems for Λ, \( \mathcal{N} \) and \( \Lambda^r \). The other rules for typing \( \Lambda^r \) are analogous to (var), (lam), (eq) of Figure 1(a) and are omitted.

\[ [M]^D = \{(\Gamma, \sigma) \mid \Gamma \vdash^D M : \sigma\} \], where the type system \( \vdash^D \) is given in Fig. 1(a).

The definition of \( [t]^D \) for \( t \in \Lambda^r \) is analogous, using the rules of Fig. 1(b). Note that \( \vdash^D \) also works for terms in \( \mathcal{N} : \bot \) is not typable, but e.g. \( \vdash^D \lambda x. \perp : (\omega \rightarrow \tau) \rightarrow \tau \).

Example 3.6 Let \( \mathcal{D} \) be any rgm. Then we have: \( [I]^D = \{\sigma \mid \sigma \simeq \tau \rightarrow \tau, \tau \in T_D\} \), \( [1]^D = \{\sigma \mid \sigma \simeq (\mu \rightarrow \tau) \rightarrow \mu \rightarrow \tau, \tau \in T_D, \mu \in \mathcal{D}\} \), \( [J]^D = \{\sigma \mid \sigma \simeq (\omega \rightarrow \tau) \rightarrow \omega \rightarrow \tau, \tau \in T_D\} \), \( [\lambda x. \Omega]^D = \{\sigma \mid \sigma \simeq (\omega \rightarrow \tau) \rightarrow \tau, \tau \in T_D\} \), \( [\Omega]^D = \emptyset \). It follows that \( [I] = [1] \) in both \( D_0 \) and \( D_* \), but \( [I]^{D_0} = [J]^{D_0} \), while \( * \in [I]^{D_*} - [J]^{D_*} \).

When \( \mathcal{D} \) is clear from the context we simply write \( \simeq, \vdash \) and \( [-] \). Note that \( \Gamma \vdash M : \sigma \) implies \( \text{dom}(\Gamma) \subseteq \text{fv}(M) \) and \( \Gamma' \vdash M : \sigma' \) for \( \Gamma \simeq \Gamma' \) and \( \sigma \simeq \sigma' \) [19].

**Theorem 3.7 (Inversion Lemma, cf. [19])** Let \( \mathcal{D} \) be an rgm.

(i) \( \Gamma \vdash x : \sigma \) entails \( \Gamma = x : \tau \) for \( \tau \simeq \sigma \),

(ii) \( \Gamma \vdash \lambda x. M : \sigma \) if and only if \( \Gamma, x : \mu \vdash M : \tau \) for some \( \mu \rightarrow \tau \simeq \sigma \),

(iii) \( \Gamma \vdash MN : \sigma \) entails \( \Gamma = \Gamma_0 \land (\bigwedge_{i=1}^n \Gamma_i) \) for some \( n \geq 0 \), \( \Gamma_0 \vdash M : \bigwedge_{i=1}^n \sigma_i \rightarrow \sigma \) and \( \Gamma_i \vdash N : \sigma_i \).

For resource \( \lambda \)-terms an analogous statement holds, where (iii) is replaced with:

(iii’) \( \Gamma \vdash [s_1, \ldots, s_n] : \sigma \) entails \( \Gamma = \Gamma_0 \land (\bigwedge_{i=1}^n \Gamma_i) \), \( \Gamma_0 \vdash t : \bigwedge_{i=1}^n \sigma_i \rightarrow \sigma \) and \( \Gamma_i \vdash s_i : \sigma_i \).

**Theorem 3.8** Let \( \mathcal{D} \) be an rgm, then for \( \Lambda \) and \( \Lambda^r \):

(i) Substitution lemma, subject reduction and subject expansion hold in \( \vdash^D \).

(ii) The interpretation \( [-]^D \) is sound with respect to \( =_\beta \).

**Proof** (i) is proved in [19] for \( \Lambda \) and in [16] for relational models of \( \Lambda^r \).

(ii) follows from (i). \( \square \)

The \( \lambda \)-theory and the order theory induced by \( \mathcal{D} \) are given by \( \text{Th}(\mathcal{D}) = \{(M, N) \mid [M] = [N]\} \) and \( \text{Th}_\leq(\mathcal{D}) = \{(M, N) \mid [M] \subseteq [N]\} \), respectively. We write \( \mathcal{D} \models M = N \) if \( (M, N) \in \text{Th}(\mathcal{D}) \), and \( \mathcal{D} \models M \leq N \) if \( (M, N) \in \text{Th}_\leq(\mathcal{D}) \).
A model $\mathcal{D}$ is $O$-inequationally fully abstract when $\mathcal{D} \models M \leq N$ if and only if\n$M \sqsubseteq^O N$, and $O$-fully abstract when $\mathcal{D} \models M = N$ if and only if $M \equiv^O N$.

**Lemma 3.9** If $\mathcal{D}$ is an extensional rgm, then $\lambda\beta\eta \subseteq \text{Th}(\mathcal{D})$.

**Proof** The equivalence between $\Gamma \vdash M : \sigma$ and $\Gamma \vdash \lambda x. Mx : \sigma$ when $x \notin \text{fv}(M)$ follows by induction on $\sigma$ using the fact that $\alpha \simeq \mu \to \tau$ for every atomic type $\alpha$. □

As a consequence, the $\lambda$-theories induced by rgms and by regular graph models are different, since no graph model is extensional. For instance, the $\lambda$-theory of $D_\omega$, the relational analogue of Scott’s $D_\infty$, is $H^*$ [15]. That is $D_\omega$ is hnf-fully abstract.

While approximation theorems for Böhm trees and idempotent intersection type systems are usually proved through reducibility techniques, the following one for $\text{SN}$ can be proved by induction on the type derivation using the subject reduction (Theorem 3.8) and the $\text{SN}$ of $\Lambda^r$ (Theorem 2.1).

**Theorem 3.10 (Approximation Theorem)** Let $M$ be a $\lambda$-term. Then

$\Gamma \vdash M : \sigma$ if and only if there exists $t \in \mathcal{T}(M)$ such that $\Gamma \vdash t : \sigma$.

Therefore $[M] = [\mathcal{T}(M)]$.

**Corollary 3.11** For all rgms $\mathcal{D}$ we have that $\mathcal{B} \subseteq \text{Th}(\mathcal{D})$. In particular $\text{Th}(\mathcal{D})$ is sensible and $[M]^\mathcal{D} = \emptyset$ for all unsolvable $\lambda$-terms $M$.

**Proof** From Theorem 3.10 we have $[M] = [\mathcal{T}(M)] = \bigcup_{t \in \mathcal{T}(M)}[t]$. By subject reduction for $\Lambda^r$ (Theorem 3.8) this is equal to $\bigcup_{t \in \mathcal{T}(M)}[\text{hnf}_\beta(t)]$, which is equal to $\bigcup_{t \in \mathcal{T}(\text{BT}(M))}[t] = [\mathcal{T}(\text{BT}(M))]$, by Theorem 2.4. Therefore, whenever $\text{BT}(M) = \text{BT}(N)$ we get $[M] = [\mathcal{T}(\text{BT}(M))] = [\mathcal{T}(\text{BT}(N))] = [N]$. □

## 4 Full Abstraction for Morris’s Observational Preorder

This section is devoted to show that every extensional rgm $\mathcal{D}$ satisfying the condition of Definition 4.1 — in particular $\mathcal{D}_*$ — is (inequationally) fully abstract with respect to Morris’s pre-order $\sqsubseteq^{nf}$. Rather than working directly with $\sqsubseteq^{nf}$, and building separating contexts, we use Levy’s notion of extensional Böhm tree

$\text{BT}^{e}(M) = \{\text{hnf}_\eta(a) \mid a \in \text{BT}(M')^*, M' \rightarrow_\eta M\}$.

Indeed, it is well known that $M \sqsubseteq^{nf} N$ exactly when $\text{BT}^{e}(M) \subseteq \text{BT}^{e}(N)$ [11] and that two $\lambda$-terms have the same extensional Böhm tree when their Böhm trees are equal up to (possibly infinitely many) $\eta$-expansions of finite depth. These trees are therefore different from Nakajima trees: for instance $I \in \text{BT}^{e}(I) - \text{BT}^{e}(J)$.

Examples of extensional Böhm trees are: $\text{BT}^{e}(1) = \text{BT}^{e}(I)$, $\text{BT}^{e}(I) = \{\perp, I, \lambda x_0.x \perp, \lambda x_0.\lambda y_0.\lambda z_0.\lambda z_1.\lambda z_2.\perp, \ldots\}$, $\text{BT}^{e}(J) = \text{BT}^{e}(I) - \{I\}$, $\text{BT}^{e}(\lambda y.x y y) = \{\perp, x \perp, \lambda y.x y y, \lambda y.x y y \perp, \ldots\}$, $\text{BT}^{e}(x \Omega) = \text{BT}^{e}(\lambda y.x y y) - \{\lambda y.x y y\}$.

Given a polarity $p \in \{+, -, \}$, we define inductively for all types $\sigma$ the relations

$\omega \in^p \sigma$ and $\omega \in^\neg p \sigma$, where $\neg p$ is the opposite polarity, as: (i) $\omega \in^\neg p \mu \to \tau$ if
\[ \mu = \omega; \ (ii) \text{ if } \omega \in^p \tau \text{ then } \omega \in^p \mu \rightarrow \tau; \ (iii) \text{ if } \omega \in^p \tau \text{ then } \omega \in^p \tau \wedge \mu \rightarrow \tau'. \]

When \( \omega \in^+ \sigma \) (\( \omega \in^- \sigma \)) we say that \( \omega \) occurs positively (negatively) in \( \sigma \). We write \( \omega \notin^+ \sigma \) (\( \omega \notin^- \sigma \)) if \( \omega \) does not occur positively (negatively) in \( \sigma \). These notions extend to intersections in the obvious way, for instance \( \omega \in^p \sigma_1 \land \cdots \land \sigma_n \) if \( \omega \in^p \sigma_i \) for some \( i \).

**Definition 4.1** An rmg \( D \) preserves \( \omega \)-polarities whenever \( \omega \in^p \sigma \) and \( \sigma \simeq \tau \) entail \( \omega \in^p \tau \), for all \( \sigma, \tau \in T_D \) and \( p \in \{+,-\} \).

For instance \( E \) and \( D_* \) preserve \( \omega \)-polarities, while \( D_\omega \) does not because \( \omega \in^+ (\omega \rightarrow \varepsilon) \rightarrow \varepsilon \simeq \varepsilon \rightarrow \varepsilon \) but \( \omega \notin^+ \varepsilon \rightarrow \varepsilon \). Note that, if an rmg \( D \) preserve \( \omega \)-polarities, then we also have that \( \omega \notin^p \sigma \) and \( \sigma \simeq \tau \) entail \( \omega \notin^p \tau \) (where \( p \in \{+,-\} \)).

**Proposition 4.2** Let \( A \) be a partial pair such that, for all \( m \in M_\varphi(A) \) and \( \alpha \in A \), \((m, \alpha) \in \text{dom}(j) \) entails that \( m \neq [] \). Then \( A \) preserves \( \omega \)-polarities.

**Lemma 4.3** Let \( M \in \Lambda \). The following are equivalent:

1. \( M \) has a normal form,
2. there is \( a \in \text{BT}(M)^* \) that does not contain \( \perp \),
3. there is \( t \in \text{nf}_\beta(\mathcal{T}(M)) \) that does not contain the empty bag \( 1 \),
4. in every rmg \( D \) preserving \( \omega \)-polarities, \( \Gamma \vdash^D M : \sigma \) for some environment \( \Gamma \) and type \( \sigma \) such that \( \omega \notin^+ \sigma \) and \( \omega \notin^- \Gamma \) (that is \( \omega \notin^- \Gamma(x) \) for all \( x \in \text{Var} \)).

**Proof** [Sketch] (i) \( i \iff ii \) is trivial and (ii) \( iiff iii \) follows from Theorem 2.4.

(iii) \( \Rightarrow iv \) One proves by induction on the \( \beta \)-normal \( t \) that \( \Gamma \vdash t : \sigma \) holds for some \( \Gamma, \sigma \) such that \( \omega \notin^+ \Gamma \) and \( \omega \notin^+ \sigma \). Then one concludes by subject expansion for \( \Lambda^* \) and the approximation theorem (Theorem 3.10).

(iv) \( \Rightarrow iii \) By the approximation theorem and subject reduction for \( \Lambda^* \) there is \( t \in \text{nf}_\beta \mathcal{T}(M) \) such that \( \Gamma \vdash t : \sigma \) is derivable for some \( \Gamma, \sigma \) satisfying \( \omega \notin^+ \Gamma \) and \( \omega \notin^+ \sigma \). Then, using Theorem 3.7 and the preservation of \( \omega \)-polarities, one proves by induction on the structure of normal form of \( t \) that it does not contain 1. \( \square \)

Notice that in the model \( D_\omega \), which does not preserve \( \omega \)-polarities, the above lemma does not hold. For instance, \( \omega \notin^+ \varepsilon \rightarrow \varepsilon \in \left[ \mathbb{J} \right]^{D_\omega} \), but \( \mathbb{J} \) is not normalizing.

In Coppo, Dezani and Zacchi’s model \( D_{cdz} \) presented in [6], there is an atomic type \( \varphi_* \) (resp. \( \varphi_\tau \)) characterizing the terms having a \( \beta \)-nf (resp. persistent \( \beta \)-nf).

In the model \( D_* \) the type \( * \) captures those \( \lambda \)-terms \( M \in \Lambda^o \) having a normal form that is “linear”. A \( \lambda \)-term \( M \) is called linear whenever: (i) every \( y \in \text{fv}(M) \) occurs once in \( M \); (ii) every subterm \( \lambda x. N \) of \( M \) is such that \( x \) occurs once in \( N \).

**Lemma 4.4** Let \( M \in \Lambda \) and \( \Gamma = x_1 : *, \ldots, x_n : * \). Then \( \Gamma \vdash^D M : * \) if and only if \( M \) has a linear \( \beta \)-normal form and \( \text{fv}(\text{nf}_\beta(M)) = \text{dom}(\Gamma) \).

We now prove the main results of the section.

**Theorem 4.5** Let \( D \) be an extensional rmg preserving \( \omega \)-polarities. The following are equivalent (for \( M, N \in \Lambda^o \)):

1. \( D \models M \leq N \),
(iii) \( M \sqsubseteq_{\text{nf}} N \),

(iii) \( \text{BT}^e(M) \subseteq \text{BT}^e(N) \).

**Proof** (\( i \Rightarrow ii \)) Suppose \([M] \subseteq [N]\) and consider a context \( C(-)\) such that \( C(M)\) has a normal form. By Lemma 4.3 there is \( \sigma \in [C(M)]\) such that \( \omega \notin \uparrow \sigma\). Since \([\_\_\_]\) is contextual we have \([C(M)] \subseteq [C(N)]\), therefore \( \sigma \in [C(N)]\) and, by applying Lemma 4.3 again, we conclude that \( C(N)\) has a normal form.

\( ii \iff iii \) See Hyland’s original paper [11], or [20] for a cleaner proof.

(iii) \( \Rightarrow i \) We have:

\[
[M] = \bigcup_{M' \rightarrow_{\eta} M}[M']
\]

by Lemma 3.9

\[
= \bigcup_{M' \rightarrow_{\eta} M}[\text{T}(M')]
\]

by Theorem 3.10

\[
= \bigcup_{M' \rightarrow_{\eta} M}[\text{inf}_\beta \text{T}(M')]
\]

by Theorem 3.8(ii) for \( \Lambda^r\)

\[
= \bigcup_{M' \rightarrow_{\eta} M}[\text{BT}(M')^*]
\]

by Theorem 2.4

\[
= \bigcup_{M' \rightarrow_{\eta} M}[\text{inf}_\eta \text{BT}(M')^*]
\]

by Lemma 3.9

\[
= [\text{BT}^e(M)]
\]

by definition of \( \text{BT}^e(M)\).

Thus \( \text{BT}^e(M) \subseteq \text{BT}^e(N)\) entails \([M] = [\text{BT}^e(M)] \subseteq [\text{BT}^e(N)] = [N]\). \( \square \)

**Corollary 4.6 (Full abstraction)** Every extensional \( \text{rgm} \ D \) respecting \( \omega\)-polarities has order-theory \( \text{Th}_{\leq}(D) = \{(M, N) \mid M \sqsubseteq_{\text{nf}} N\} \) and \( \lambda\)-theory \( \text{Th}(D) = \mathcal{H}^+\).

## 5 Extensional Taylor Expansion and \( \eta\)-Trees

We introduce the notion of *extensional Taylor expansion* \( \mathcal{T}^\eta(M)\) of a \( \lambda\)-term \( M\) and prove that it is equal to the Taylor expansion of the extensional Böhm tree of \( M\) (Theorem 5.15). This result is the analogue of Theorem 2.4. As a byproduct, we obtain a new syntactical characterization of \( \equiv_{\text{nf}}\) (Corollary 5.17).

For technical reasons, we work with an alternative notion of extensional Böhm tree of \( M\), that will be denoted by \( \text{BT}^\eta(M)\). Rather than producing a set of \( \eta\)-normal approximants, \( \text{BT}^\eta(-)\) gives an actual (possibly infinite) \( \eta\)-normal tree.

The \( \eta\)-normal form \( \eta(T)\) of a Böhm tree \( T\) is defined coinductively: \( \eta(\bot) = \bot \) and

\[
\eta\left( \lambda x_1 \ldots x_n.\ y \atop T_1 \ldots \atop T_m \right) = \begin{cases} \eta\left( \lambda x_1 \ldots x_{n-1}.y \atop T_1' \ldots \atop T_{m-1} \right) & \text{If } x_n \notin \text{fv}(yT_1 \cdots T_{m-1}) \\
\lambda x_1 \ldots x_n.\ y & \text{otherwise} \\
\eta(T_1) \ldots \eta(T_m) & \text{and } T_m \rightarrow_{\eta} x_n, \text{ i.e. it is finite} \\
& \text{and } T_m \in \mathcal{N}, \text{ i.e. it is finite} \\
\end{cases}
\]

Therefore, we define the Böhm \( \eta\)-tree \( \text{BT}^\eta(M)\) of a \( \lambda\)-term \( M\) as \( \eta(\text{BT}(M))\).

Examples of Böhm \( \eta\)-trees are: \( \text{BT}^\eta(\text{J}) = \text{BT}(\text{J})\), \( \text{BT}^\eta(\lambda y.xyy) = \lambda y.xyy\), \( \text{BT}^\eta(\lambda xy_1x_2. x(\lambda z_1.y_1(\lambda z_2.z_1(\lambda z_3.z_2z_3)y_2))y_2) = \text{BT}^\eta(\text{I}) = \text{I}\), and \( \text{BT}^\eta(\lambda y.x\bot y) = x\bot\).
The notions of $\text{BT}^\eta(-)$ and $\text{BT}^\epsilon(-)$ are equivalent in the sense that, for all $M, N \in \Lambda$, $\text{BT}^\epsilon(M) = \text{BT}^\epsilon(N)$ if and only if $\text{BT}^\eta(M) = \text{BT}^\eta(N)$ [23,14]. On the other hand, $\text{BT}^\epsilon(M) \subseteq \text{BT}^\epsilon(N)$ is not equivalent to $\text{BT}^\eta(M) \preceq \text{BT}^\eta(N)$. E.g. $\text{BT}^\epsilon(x \perp) \subseteq \text{BT}^\epsilon(\lambda y.xxx)$ but $\text{BT}^\eta(x \perp) = x \perp \not\preceq \lambda y.xxx = \text{BT}^\eta(\lambda y.xxx)$.

### 5.1 Extensional Taylor Expansion

In order to obtain the analogue of Ehrhard and Regnier’s Theorem 2.4 in the extensional setting, the extensional Taylor expansion of $M$ should be the $\eta$-normal form of $\text{nf}_\beta T(M)$, just like $\text{BT}^\eta(M)$ is the $\eta$-normal form of $\text{BT}(M)$.

The problem is that defining an $\eta$-reduction on $\mathcal{P}(\text{nf}_\beta(\Lambda'))$ is no easy task. Consider for instance the naïve definition $\rightarrow^{\eta} = \bigcup_{k \geq 0}(\rightarrow^{\eta_k})$ where $\lambda x.t[x^k] \rightarrow^{\eta_k} t$ if $x \notin \text{fv}(t)$. This correctly reduces $T(\lambda y.xy) = \{\lambda y.x[y^k] \mid k \geq 0\}$ to $\{x\}$, but the fact that $\lambda y.x1 \rightarrow^{\eta_0} x$ is a problem, since $\lambda y.x1$ also belongs to $T(\lambda y.x\Omega)$, whereas $x \notin T(\text{nf}_\eta(\lambda y.x\Omega)) = \{\lambda y.x1\}$. Similarly, $\lambda y.x1[y]$ as an element of $T(\lambda y.xzy)$ is supposed to $\eta$-reduce to $x1$, while as an element of $T(\lambda y.xyy)$ should be $\eta$-normal.

These examples reveal that, while the $\beta$-reduction of $T(M)$ can be performed locally by reducing each term individually, the $\eta$-reduction of $\text{nf}_\beta T(M)$ must be a global operation, that considers the whole set of terms before deciding whether a term should reduce or not. Rather than defining an infinitary rewriting system handling countably many terms, we prefer to divide the problem of computing the $\eta$-normal form of $T(M)$ into two phases:

(i) we first define a labeling $\mathcal{L}(-)$ on the terms $t \in T(M)$ as a global operation annotating on the empty bags 1 occurring in $t$:

- whether they “come from” a finite $\eta$-expansion of some variable $y$; for instance $\lambda y.x1 \in T(\lambda y.x(\lambda z.yz))$ should be labeled as $\lambda y.x1^{\eta(y)}$;
- the set of free variables that were forgotten by taking 1 in the Taylor expansion; for instance $\lambda y.x1[y] \in T(\lambda y.xyy)$ should be labeled as $\lambda y.x1[y]$.

(ii) We then define a local reduction $\rightarrow^{\eta'}_\beta$ on $\mathcal{L}(\text{nf}_\beta T(M))$ that exploits this extra-information annotated to perform the $\eta$-reduction only when it is safe.

The definition of the labeling $\mathcal{L}$ (Definition 5.1) relies on a certain homogeneity exhibited by the structure of the resource terms in $\text{nf}_\beta T(M)$. As shown in [4], this homogeneity relies on a definedness relation $\preceq$ between normal resource terms:

$$\lambda x_1 \ldots x_n.y \preceq \lambda x_1 \ldots x_n.y \quad \frac{t \preceq t' \quad b \preceq b'}{tb \preceq t'b'} \quad \frac{1 \preceq b}{\exists t' \in b' \forall t \in b, t \preceq t'}$$

The relation $\preceq$ is not a preorder since it is transitive, but not reflexive. For instance, $x[y1[y],y[y1]] \not\preceq x[y1[y],y[y1]]$, since $y1[y] \not\preceq y[y1] \not\preceq y1[y]$. See the discussion after Definition 9 in [4] for more properties of this relation, and examples. Notice that all singletons $\{\lambda x_1 \ldots x_n.y\} \ (n \geq 0)$ are ideals with respect to $\preceq$.

By Lemma 12 in [4], every ideal $S$ has one of the following shapes: $\{x\}$, $\lambda x. T$, $\text{TB}$ for some ideals $T$ and $B$. Therefore, the following definition is sound.
Definition 5.1 Let $S \subseteq \text{nf}_\beta(\Lambda^r)$ be an ideal with respect to $\leq$ and $t \in S$. The labeled term $L(t, S)$ is defined as follows:

$$L(x, \{\}) = x, \quad L(\lambda x.t, \lambda x.T) = \lambda x.L(t, T), \quad L(tb, TB) = L(t, T)L(b, B),$$
$$L([t_1, \ldots, t_k], B) = [L(t_1, \cup B), \ldots, L(t_k, \cup B)], \text{ for } k > 0$$

$$L(1, B) = \begin{cases} 1^n_{\eta}(x) & \text{if there exists } t' \in \cup B \text{ such that } t' \rightarrow_{\eta'} x, \quad (\bullet) \\ 1^{f_v(B)} & \text{otherwise.} \end{cases}$$

where $\rightarrow_{\eta'}$ is $\lambda x.t[x^{k+1}] \rightarrow_{\eta'} t$ when $x \notin f_v(t)$. We set $L(S) = \{L(t, S) \mid t \in S\}$.

Given a labelled term $t$, we write $\rightarrow^* t$ for the term obtained by erasing all its labels.

The labeling $L(-)$ can be always applied to $\text{nf}_\beta T(M)$ thanks to the following.

Proposition 5.2 [4, Lemma 23] Let $M \in \Lambda$. Then $\text{nf}_\beta T(M)$ is an ideal w.r.t. $\leq$.

Remark 5.3 The definition of $L(t, S)$ will be only used when $S$ is the $\beta$-normal of a Taylor expansion. Under this hypothesis, the case $L(1, B)$ is applied when $\cup B = T(M)$ for some $\beta$-normal $M \in \Lambda$ and Condition $(\bullet)$ becomes “there is $t \in T(M)$ such that $t \rightarrow_{\eta'} x$” which holds exactly when $M \rightarrow_\eta x$.

For example, for $t = \lambda y.x11$ and $S = \text{nf}_\beta T(\lambda y.x1\Omega y) = \{\lambda y.x1[y^n] \mid n \geq 0\}$ we have $L(t, S) = \lambda y.L(x, \{\})L(1, \{\})L(\{\{y^k \mid k \geq 0\}\}) = \lambda y.x1^{\varnothing}1^y(y)$. While $L(\lambda y.x11, \text{nf}_\beta T(\lambda y.xyy)) = \lambda y.x1^y_\eta(y)1^y_\eta(y)$. Thus $L(T(\lambda y.xyy)) = \{\lambda y.x1^y_\eta(y)1^y_\eta(y)\}$, which is always $\eta$-normal.

The definition of the set $f_v(t)$ of free variables of a labeled term $t$ is analogous to the one of $f_v(t)$, except for the clauses $f_v(1^n_{\eta}(x)) = \{x\}$ and $f_v(1^\eta(x)) = \{x\}$.

Remark 5.4 Given $T = BT(M), x \in f_v(T)$ iff $x \in \tilde{f}_v(t)$ for every $t \in L(T(T))$.

Definition 5.5 The reduction $\rightarrow_{\eta'}$ on labelled $\beta$-normal resource terms, is the contextual closure of $\cup_{n \in \mathbb{N}}(\rightarrow_{\eta'_n})$ where $\rightarrow_{\eta'_n}$ is defined as follows:

$$(\eta'_0) \lambda x.1^x_n(x) \rightarrow_{\eta'_0} t \text{ if } x \notin \tilde{f}_v(t), \quad (\eta'_{n+1}) \lambda x.t[x^{n+1}] \rightarrow_{\eta'_{n+1}} t \text{ if } x \notin \tilde{f}_v(t).$$

For example, we have $L(\lambda y.x1[y], \text{nf}_\beta T(\lambda y.xyy)) = \lambda y.x1^z_{\eta'(y)}[y] \rightarrow_{\eta'} x1^z_{\eta'(y)},$ while $L(\lambda y.x1[y], \text{nf}_\beta T(\lambda y.xyy)) = \lambda y.x1^y_{\eta(y)}1^y_{\eta(y)}y$, which is already $\eta'$-normal.

Lemma 5.6 The reduction $\rightarrow_{\eta'}$ is SN and confluent.

Proof The reduction $\rightarrow_{\eta'}$ is SN since the size of the term decreases. It is moreover weakly confluent, and therefore confluent by Newman’s lemma.

Definition 5.7 The extensional Taylor expansion of a $\lambda$-term $M$ is given by:

$$T^\eta(M) = \text{nf}_{\eta'} L(\text{nf}_\beta T(M))^\eta$$

In the definition above, $\beta$- and $\eta'$-reductions are separated because the reduction $\beta \cup \eta'$ is not confluent: for instance $\lambda x.1[x, x] \rightarrow_{\eta'} 1$ while $\lambda x.1[x, x] \rightarrow_\beta \emptyset$. 

259
5.2 Eta-Reduction on Böhm Approximants

We now provide the technical tools that will be used to prove Theorem 5.15. By Theorem 2.4, it is enough to prove that \( T(B^\eta(M)) \) is equal to \( \text{nf}_{\eta}(L(T(BT(M))))^\eta \). The difficulty lies in that \( BT^\eta(M) \), which is the \( \eta \)-normal form of \( BT(M) \), is defined coinductively on \( BT(M) \), while the \( \eta \)-reduction of \( T(BT(M)) \) works on a set of (labeled) resource terms coming from the finite approximants in \( BT(M)^* \). Therefore, as an intermediate step, we define the \( \eta \)-normal form of the set \( BT(M)^* \) mimicking what we did in Subsection 5.1 for sets of resource terms. In particular, even in this framework the \( \eta \)-reduction must be a global operation; therefore, we introduce a labeling on finite approximants in the spirit of Definition 5.1.

Given \( M \subseteq N \), \( M \downarrow \) denotes its downward closure \( \{ a \in N \mid \exists b \in M \text{, } a \leq_b b \} \).

When \( M \) is an ideal, we have that \( M = M \downarrow \) and all its elements have a similar syntactic structure, except for \( \bot \). We adopt for sets \( M \) of approximants the same syntactic sugar we used for \( P(N^\eta) \), by extending all the constructors of the grammar of \( N \) as pointwise operations on \( P(N) \). For instance the ideal \( BT(Jx)^* \) can be written as \( \{ \lambda z_0.x(BT(Jz_0)^*) \} \downarrow = \lambda z_0.x(BT(Jz_0)^*) \cup \{ \bot \} \).

**Definition 5.8** Let \( M \subseteq N \) be an ideal w.r.t. \( \leq \) and \( a \in M \). Define \( E(a, M) \) as:

\[
E(x, \{ x \} \downarrow) = x, \quad E(\lambda x.a, (\lambda x.M) \downarrow) = \lambda x.E(a, M \downarrow),
\]

\[
E(ac, (MN) \downarrow) = E(a, M \downarrow)E(c, N),
\]

\[
E(\bot, M) = \begin{cases} \bot^x & \text{if there exists a } \bot\text{-free } a \in M \text{ such that } a \rightarrow^\eta x, \quad (\circ) \\ \bot^{fv(M)} & \text{otherwise.} \end{cases}
\]

We extend the definition to \( M \) by setting \( E(M) = \{ E(a, M) \mid a \in M \} \).

Notice that in the case \( (MN) \downarrow \) above, the set \( N \) is already downward closed. As \( BT(M)^* \) is an ideal for every \( M \in \Lambda \), we can always compute \( L(BT(M)^*) \).

Condition (\( \circ \)) is then equivalent to check that \( M = BT(M')^* \) for some \( M' \rightarrow^\eta x \).

As we did for resource terms, we speak of labeled approximants \( a \), we define the set \( \text{fv}(a) \) by adding the clauses \( \text{fv}(\bot^x) = \{ x \} \) and \( \text{fv}(\bot x) = \{ x \} \), and we write \( \text{fv}(a) \) for the term obtained from \( a \) by erasing all its labels.

**Remark 5.9** Given \( T = BT(M) \), \( x \in \text{fv}(T) \) iff \( x \in \text{fv}(t) \) for every \( t \in E(T^*) \).

**Definition 5.10** The reduction \( \rightarrow_{\eta^*} \) on labeled approximants is defined as:

\[
\lambda x.a.\bot^x \rightarrow_{\eta^*} a, \text{ if } x \notin fv(a), \quad \lambda x.ax \rightarrow_{\eta^*} a, \text{ if } x \notin \text{fv}(a).
\]

It is easy to check that also \( \rightarrow_{\eta^*} \) is strongly normalizing and confluent.

After a technical lemma, we show that the \( \eta^* \)-reduction on \( E(BT(M)) \) computes exactly the finite approximants of the co-inductively defined tree \( BT^\eta(M) \). Given two sets of terms \( X, Y \) and a reduction \( \rightarrow_r \) we write \( X \Rightarrow_r Y \) if for all \( t_1 \in X \) there is \( t_2 \in Y \) such that \( t_1 \rightarrow_r t_2 \) and for all \( t_2 \in Y \) there is \( t_1 \in X \) such that \( t_1 \rightarrow_r t_2 \).
Lemma 5.11 Let $T = \lambda x y z T_1 \cdots T_{k+1}$ be a Böhm tree such that $T_{k+1}$ is finite, $T_{k+1} \rightsquigarrow_\eta y$ and $y \notin \text{fv}(zT_1 \cdots T_k)$. Then $E(T^*) \Rightarrow_{\eta^*} E((\lambda x z T_1^* \cdots T_k^*)) \downarrow$.

Proposition 5.12 For all $M \in \Lambda$, we have $B\bar{T}^\eta(M)^* = \Gamma \text{nf}_{\eta^*} E(B\bar{T}(M)^*)^\gamma$.

Proof [Sketch] One proceeds by co-induction on $B\bar{T}(M)$ using Lemma 5.11.

5.3 A Taylor-Based Characterization of Morris’s Equivalence

Now that the technical tools for proving the main result of the section are finally in place, we are able to prove that the extensional Taylor expansion of a $\lambda$-term $M$, actually captures the Taylor expansion of $B\bar{T}^\eta(M)$.

We first need the following technical results, then we show a sort of commutation between the $\eta^f$-normalization and the Taylor expansion.

Lemma 5.13 Let $T = \lambda x y z T_1 \cdots T_{k+1}$ be a Böhm tree such that $T_{k+1}$ is finite, $T_{k+1} \rightsquigarrow_\eta y$ and $y \notin \text{fv}(zT_1 \cdots T_k)$. Then $L(T(T)) \Rightarrow_{\eta^f} L(T(\lambda x z T_1 \cdots T_k))$.

Proposition 5.14 For all $M \in \Lambda$, $T(\Gamma \text{nf}_{\eta^f} E(B\bar{T}(M)^*)^\gamma) = \Gamma \text{nf}_{\eta^f} L(T(B\bar{T}(M)))^\gamma$.

Proof [Sketch] By coinduction on $B\bar{T}(M)$, applying Lemma 5.13.

We can finally prove the main result of the section.

Theorem 5.15 For every $\lambda$-term $M$, $\bar{T}^\eta(M) = T(B\bar{T}^\eta(M))$.

Proof Collecting the results above, we have the following chain of equalities:

$$
\bar{T}^\eta(M) = \Gamma \text{nf}_{\eta^f} L(\text{nf}_\beta T(M))^\gamma \quad \text{by Definition 5.7}
$$

$$
= \Gamma \text{nf}_{\eta^f} L(T(B\bar{T}(M)))^\gamma \quad \text{by Theorem 2.4}
$$

$$
= T(\Gamma \text{nf}_{\eta^f} E(B\bar{T}(M)^*)^\gamma) \quad \text{by Prop. 5.14}
$$

$$
= T(B\bar{T}^\eta(M)^*) \quad \text{by Prop. 5.12}
$$

Corollary 5.16 For all $M, N \in \Lambda$, we have $B\bar{T}^\eta(M)^* \subseteq B\bar{T}^\eta(N)^*$ if and only if $\bar{T}^\eta(M) \subseteq \bar{T}^\eta(N)$.

Proof $(\Rightarrow)$ Let $t \in \bar{T}^\eta(M)$. Then there is $a \in B\bar{T}^\eta(M)^*$ such that $t \in T(a)$. Since $B\bar{T}^\eta(M)^* \subseteq B\bar{T}^\eta(N)^*$, we have that $a \in B\bar{T}^\eta(N)^*$. So $t \in T(B\bar{T}^\eta(N))$ and we get from Theorem 5.15 that $t \in \bar{T}^\eta(N)$.

$(\Leftarrow)$ Let $a \in B\bar{T}^\eta(M)^*$. Then by Theorem 5.15 $T(a) \subseteq T(B\bar{T}^\eta(M)) = \bar{T}^\eta(M) \subseteq \bar{T}^\eta(N)$. Since $\bar{T}^\eta(N) = T(B\bar{T}^\eta(N))$ holds still by Theorem 5.15, we have that $T(a) \subseteq T(B\bar{T}^\eta(N))$. From Lemma 2.3 we conclude that $a \in B\bar{T}^\eta(N)^*$.

A further corollary is that the notion of extensional Taylor expansion provides an alternative characterization of Morris’s equivalence.

Corollary 5.17 For $M, N \in \Lambda$, we have $M \equiv_{\text{nf}} N$ if and only if $\bar{T}^\eta(M) = \bar{T}^\eta(N)$. 261
We have the following chain of equivalences: By [23] \( M \equiv^nf N \) if and only if \( BT^0(M) = BT^0(N) \), that is \( BT^0(M)^* = BT^0(N)^* \). By Corollary 5.16 this holds if and only if \( T^0(M) = T^0(N) \) does.

Thanks to Henk Barendregt, Mariangiola Dezani, Thomas Ehrhard, Stefano Guerrini, Michele Pagani for stimulating discussions and the reviewers for the comments.

References


A Technical Appendix

This technical appendix is devoted to provide some proofs that were omitted or just sketched in the article.

A.1 Omitted proofs of Section 2

2.3 Let \( a \in N \) and \( \mathcal{M} \in \Lambda \), then \( \mathcal{T}(a) \subseteq \mathcal{T}(\mathcal{BT}(M)) \) entails \( a \in \mathcal{BT}(M)^* \).

Proof. By structural induction on \( a \).

Case \( a = \perp \) and \( \mathcal{T}(a) = \emptyset \subseteq \mathcal{T}(\mathcal{BT}(M)) \). Then it is trivial since \( \perp \in \mathcal{BT}(M) \).

Case \( a = \lambda x.ya_1 \cdots a_k \) and \( \mathcal{T}(a) = \bigcup_{n_1, \ldots, n_k \geq 0} \lambda x.y[\mathcal{T}(a_1)^{n_1}] \cdots [\mathcal{T}(a_k)^{n_k}] \subseteq \mathcal{T}(\mathcal{BT}(M)) \). Then \( M \rightarrow_\beta \lambda x.yN_1 \cdots N_k \) for some \( N_1, \ldots, N_k \in \Lambda \) such that \( \mathcal{T}(a_i) \subseteq \mathcal{T}(\mathcal{BT}(N_i)) \). By induction hypothesis \( a_i \in \mathcal{BT}(N_i)^* \) for all \( 1 \leq i \leq k \) and we conclude that \( \lambda x.ya_1 \cdots a_k \in \mathcal{BT}(M)^* \). \( \square \)

A.2 Omitted proofs of Section 3

3.10 (Approximation Theorem) Let \( M \) be a \( \lambda \)-term. Then \( \Gamma \vdash M : \mathcal{\sigma} \) if and only if there exists \( t \in \mathcal{\mathcal{T}(M)} \) such that \( \Gamma \vdash t : \mathcal{\sigma} \).

Proof. \(( \Rightarrow \)\) The proof is by induction on a derivation of \( \Gamma \vdash M : \mathcal{\sigma} \). We proceed by case analysis on the last rule applied in the derivation.

Case \( \text{var} \). We have \( x : \mathcal{\sigma} \vdash x : \mathcal{\sigma} \) using the rule (var). This case is trivial since \( \mathcal{T}(x) = \{x\} \).

Case \( \text{lam} \). We have \( \Gamma \vdash \lambda x.\mathcal{N} : \mathcal{\sigma} \) using the rule (lam). By Theorem 3.7(ii), we have that \( \Gamma, x : \mathcal{M} \vdash \mathcal{N} : \tau \) for some \( \mathcal{M} \rightarrow \tau \simeq \mathcal{\sigma} \). By IH, there exists \( \mathcal{t'} \in \mathcal{T}(\mathcal{N}) \) such that \( \Gamma, x : \mathcal{M} \vdash \mathcal{t'} : \tau \). Therefore, \( \lambda x.\mathcal{t'} \in \mathcal{T}(\lambda x.\mathcal{N}) \) and

\[
\frac{\Gamma, x : \mathcal{M} \vdash \mathcal{t'} : \tau \quad \text{(lam)}}{\Gamma \vdash \lambda x.\mathcal{t'} : \mathcal{\sigma}} \quad \text{(eq)}
\]

Case \( \text{app} \). We have \( \Gamma \vdash \mathcal{NP} : \mathcal{\sigma} \) using the rule (app). By Theorem 3.7(iii), there is a decomposition \( \Gamma = \Gamma_0 \land (\land_{i=1}^n \Gamma_i) \) for some \( n \geq 0 \), such that \( \Gamma_0 \vdash \mathcal{N} : \land_{i=1}^n \mathcal{\sigma}_i \rightarrow \mathcal{\sigma} \) and \( \Gamma_i \vdash \mathcal{P} : \mathcal{\sigma}_i \). By IH, there exists \( s \in \mathcal{T}(\mathcal{N}) \) such that \( \Gamma_0 \vdash s : \land_{i=1}^n \mathcal{\sigma}_i \rightarrow \mathcal{\sigma} \), and there exist \( t_1, \ldots, t_n \in \mathcal{T}(\mathcal{P}) \) such that \( \Gamma_i \vdash t_i : \mathcal{\sigma}_i \).

Therefore we have that \( s[t_1, \ldots, t_n] \in \mathcal{T}(\mathcal{NP}) \) and:

\[
\frac{\Gamma_0 \vdash s : \land_{i=1}^n \mathcal{\sigma}_i \rightarrow \mathcal{\sigma} \quad \Gamma_i \vdash t_i : \mathcal{\sigma}_i \quad \forall i \in \{1, \ldots, n\}}{\Gamma \vdash s[t_1, \ldots, t_n] : \mathcal{\sigma}}
\]

Case \( \text{eq} \). Let \( \Gamma \vdash \mathcal{M} : \mathcal{\sigma} \) using the rule (eq). Then \( \Gamma \vdash \mathcal{M} : \tau \) for some \( \tau \simeq \mathcal{\sigma} \).

By IH there exists \( t \in \mathcal{T}(\mathcal{M}) \) such that \( \Gamma \vdash t : \tau \). By applying (eq) we derive \( \Gamma \vdash t : \mathcal{\sigma} \).

This concludes the left-to-right implication.
Therefore we derive:

\[
\text{Case var. } \text{We have } x : \sigma \vdash x : \sigma \text{ and } x \in \mathcal{T}(M) \text{ which entails } M = x \text{ by definition of the Taylor expansion. This case is therefore trivial.}
\]

\[
\text{Case app. } \text{We have } s[t_1, \ldots, t_n] \in \mathcal{T}(M) \text{ such that } \Gamma \vdash s[t_1, \ldots, t_n] : \sigma. \text{ By Theorem 3.7(iii)}, \text{ we get the decomposition } \Gamma = \Gamma_0 \land (\land_{i=1}^n \Gamma_i) \text{ and the typing assignments } \Gamma_0 \vdash t : \land_{i=1}^n \sigma_i \rightarrow \sigma \text{ and } \Gamma_i \vdash s_i : \sigma_i. \text{ By definition of Taylor expansion, if } s[t_1, \ldots, t_n] \in \mathcal{T}(M) \text{ then } M = NP \text{ for some } N, P \in \Lambda \text{ such that } s \in \mathcal{T}(N) \text{ and } t_1, \ldots, t_n \in \mathcal{T}(P). \text{ By IH, } \Gamma_0 \vdash N : \land_{i=1}^n \sigma_i \rightarrow \sigma \text{ and } \Gamma_i \vdash P : \sigma_i \text{ for all } i \in \{1, \ldots, n\}. \text{ Therefore we derive:}
\]

\[
\frac{\Gamma_0 \vdash N : \land_{i=1}^n \sigma_i \rightarrow \sigma \quad \Gamma_i \vdash P : \sigma_i \quad \forall i \in \{1, \ldots, n\}}{\Gamma \vdash NP : \sigma} \quad \text{(app)}
\]

\[
\text{Case lam. } \text{We have } \lambda x.t \in \mathcal{T}(M) \text{ such that } \Gamma \vdash \lambda x.t : \sigma. \text{ By definition of Taylor expansion, } \lambda x.t \in \mathcal{T}(M) \text{ entails } M = \lambda x.N \text{ for some } N \in \Lambda \text{ such that } t \in \mathcal{T}(N). \text{ By Theorem 3.7(ii), one gets } \Gamma, x : \mu \vdash t : \tau \text{ for some } \mu \rightarrow \tau \simeq \sigma. \text{ By IH, we have } \Gamma, x : \mu \vdash N : \tau. \text{ Therefore, we can derive}
\]

\[
\frac{\Gamma, x : \mu \vdash N : \tau}{\Gamma \vdash \lambda x.N : \mu \rightarrow \tau} \quad (1\text{am})
\]

\[
\frac{\mu \rightarrow \tau \simeq \sigma}{\Gamma \vdash \lambda x.N : \sigma} \quad \text{(eq)}
\]

\[
\text{Case eq. } \text{Let } t \in \mathcal{T}(M) \text{ and suppose } \Gamma \vdash t : \sigma \text{ comes from } \Gamma \vdash t : \tau \text{ by (eq). By IH, we have } \Gamma \vdash M : \tau. \text{ By applying (eq) we derive } \Gamma \vdash N : \sigma. \quad \square
\]

A.3 Omitted proofs of Section 4

We recall the definition of “\(\omega\) occurs positively/negatively in a type \(\sigma\)”.

**Definition A.1** The relations \(\omega \in^p \sigma\) for \(p \in \{+, -\}\) are defined as follows:

(i) \(\omega \in^+ \mu \rightarrow \sigma\) for any type \(\sigma\) and intersection \(\mu\) such that \(\mu = \omega\);

(ii) if \(\omega \in^p \sigma\) then \(\omega \in^p \mu \rightarrow \sigma\) for any intersection \(\mu\);

(iii) if \(\omega \in^p \sigma\) then \(\omega \in^- \sigma \land \mu \rightarrow \tau\) for any types \(\sigma, \tau\) and intersection \(\mu\).

Remark that the condition \(\mu = \omega\) in (i) is non-trivial, since equality = between types includes the neutrality of \(\omega\). For instance \(\omega \in^- \omega \land \omega \rightarrow \sigma\) as \(\omega \land \omega = \omega\).

4.2 Let \(\mathcal{A}\) be a partial pair such that, for all \(m \in \mathcal{M}_t(A)\) and \(\alpha \in A\), \((m, \alpha) \in \text{dom}(j)\) entails that \(m \neq []\). Then \(\mathcal{A}\) preserves \(\omega\)-polarities.

**Proof** We perform an induction loading and prove that, for all type \(\sigma, \tau \in \text{dom}(\mathcal{A})\) and \(p \in \{+, -\}\): if \(\omega \in^p \sigma\) and \(\tau \simeq^A \sigma\) then \(\tau^* \notin A\) and \(\omega \in^p \tau\). In the rest of the proof we will just write \(\simeq\) for \(\simeq^A\).

We proceed by induction on the definition of \(\omega \in^p \sigma\).

**Case (i).** Suppose that \(\omega \in^p \sigma\) because \(p = -\) and \(\sigma = \omega \rightarrow \gamma\), then we need to prove that \(\omega \in^- \tau\), for any \(\tau\) such that \(\tau \simeq \sigma\), that is such that \(\tau^* = \sigma^*\). By
definition, we have:
\[ \sigma^* = (\omega \to \gamma)^* = \overline{J}([], \gamma^*) = ([], \gamma^*) \]
where the last equality follows from Definition 3.2 and the hypothesis that \( ([], \gamma^*) \notin \text{dom}(j) \). From \( \tau^* = ([], \gamma^*) \) we get that \( \tau^* \notin A \) since \( A \) does not contain any pair, and this entails that also \( \tau \) cannot be atomic.

Suppose therefore \( \tau = \mu \to \delta \), then we have \( \tau^* = (\mu \to \delta)^* = \overline{J}(\mu^*, \delta^*) = \overline{J}(([], \gamma^*)) = \sigma^* \). From the injectivity of \( \overline{J} \), we get that \( \mu^* = [] \) and \( \delta^* = \gamma^* \), so \( \tau = \omega \to \delta \) and \( \omega \in \neg \tau \).

**Case (ii).** Suppose that \( \omega \in^p \sigma \) because \( \sigma = \mu \to \gamma \) and \( \omega \in^p \gamma \). Then
\[ \sigma^* = (\mu \to \gamma)^* = \overline{J}(\mu^*, \gamma^*) = \tau^*. \]
From \( \omega \in^p \gamma \), \( \gamma \simeq \gamma \) and the induction hypothesis, we get that \( \gamma^* \notin A \) and therefore \( (\mu^*, \gamma^*) \notin \text{dom}(j) \). By Definition 3.2, we have that \( \overline{J}(\mu^*, \gamma^*) = (\mu^*, \gamma^*) \), and since this is equal to \( \tau^* \), we get \( \tau = \nu \to \delta \) for some \( \nu, \delta \). From \( \overline{J}(\mu^*, \gamma^*) = \overline{J}(\nu^*, \delta^*) \) and the injectivity of \( \overline{J} \) we get that \( \mu^* = \nu^* \) and \( \gamma^* = \delta^* \).

From \( \omega \in^p \gamma \) and \( \delta \simeq \gamma \) we get, by induction hypothesis, that \( \omega \in^p \delta \) and therefore \( \omega \in^p \nu \to \delta = \tau \).

**Case (iii).** Suppose that \( \omega \in^p \sigma \) because \( \sigma = \gamma_1 \land \mu \to \gamma_2 \) and \( \omega \in \neg \gamma_1 \). From \( \gamma_1 \land \mu \to \gamma_2 \simeq \tau \), we get
\[ (\gamma_1 \land \mu \to \gamma_2)^* = \overline{J}([\gamma_1^*] + \mu^*, \gamma_2^*) = \tau^*. \]
Suppose, by the way of contradiction, that \( \tau \) is an atomic type \( \alpha \). Then, we have \( \overline{J}([\gamma_1^*] + \mu^*, \gamma_2^*) = \alpha \) which implies, by Definition 3.2, that \( ([\gamma_1^*] + \mu^*, \gamma_2^*) \in \text{dom}(j) \subseteq \mathcal{M}_I(A) \times A \). In particular, we get \( \gamma_1^* \in A \), which is impossible since \( \omega \in \neg^p \gamma_1 \) and \( \gamma_1 \simeq \gamma_1 \), so by the induction hypothesis we conclude that \( \gamma_1^* \) is not atomic.

So, \( \tau = \nu \to \delta_2 \), and \( (\nu \to \delta_2)^* = \overline{J}(\nu^*, \delta_2^*) = \overline{J}([\gamma_1^*] + \mu^*, \gamma_2^*) \). Since \( \overline{J} \) is injective, \( \nu^* = [\gamma_1^*] + \mu^* \) and \( \delta_2^* = \gamma_2^* \). Therefore, \( \nu = \delta_1 \land \nu' \) such that \( \delta_1^* = \gamma_1^* \) and \( \nu'^* = \mu^* \).
Since \( \omega \in \neg^p \gamma_1 \) and \( \gamma_1 \simeq \delta_1 \), by III we get \( \omega \in \neg^p \gamma_1 \) and we conclude that \( \omega \in^p \tau \). \( \square \)

For convenience, we present Lemma 4.3 with an additional equivalent sentence (iii-bis), which is an intermediate step between (iii) and (iv).

4.3 Let \( M \in \Lambda \). The following are equivalent:
(i) \( M \) has a normal form,
(ii) there is \( a \in \text{BT}(M)^* \) that does not contain \( \bot \),
(iii) there is \( t \in \text{nf}_\beta(\mathcal{T}(M)) \) that does not contain the empty bag \( 1 \),
(iii-bis) in every \( \text{rgm} \ D \) preserving \( \omega \)-polarities, \( \Gamma \vdash^D t : \sigma \) for some \( t \in \text{nf}_\beta \mathcal{T}(M) \), environment \( \Gamma \) and type \( \sigma \) such that \( \omega \notin^+ \sigma \) and \( \omega \notin^\bot \Gamma \), that is \( \omega \notin^{+ \bot} \Gamma(x) \) for all \( x \in \text{Var} \).
(iv) in every \( \text{rgm} \ D \) preserving \( \omega \)-polarities, \( \Gamma \vdash^D M : \sigma \) for some environment \( \Gamma \) and type \( \sigma \) such that \( \omega \notin^+ \sigma \) and \( \omega \notin^\bot \Gamma \), that is \( \omega \notin^{+ \bot} \Gamma(x) \) for all \( x \in \text{Var} \).

**Proof** \( (i \iff ii) \) is trivial.

265
Consider then the case $k > \beta$ written as $\Gamma$. A decomposition $\Gamma = \Gamma \lambda x.t$ occur in $t'$ for some context $\Gamma$ expansion. We proceed by structural induction on $t$.

**Case** $t = \lambda x.t'$ where $t'$ is $\beta$-normal. By induction hypothesis, $\Gamma' \vdash t' : \tau$ holds for some context $\Gamma'$ and type $\tau$ such that $\omega \not\in \Gamma'$ and $\omega \not\in \Gamma^+ \tau$. Note that $\Gamma'$ can be written as $\Gamma, x : \mu$ for some $\Gamma$ and $\mu$, therefore we can derive:

$$\Gamma, x : \mu \vdash t' : \tau \quad (\text{lam})$$

From $\omega \not\in \Gamma'$ we get that $\omega \not\in \Gamma$ and $\omega \not\in \mu$, which entails $\omega \not\in \mu \rightarrow \tau$.

**Case** $t = y b_1 \cdots b_k$, for some $k \geq 0$, and each $b_i = [s_{i,1}, \ldots, s_{i,n_i}]$ (for $n_i \geq 0$) only contains $\beta$-normal terms. By induction hypothesis, there are environments $\Gamma_{ij}$, and types $\tau_{ij}$, such that $\omega \not\in \Gamma_{ij}$ and $\omega \not\in \mu_i$ and therefore $\omega \not\in \Gamma_{ij}$. From this, and the hypotheses that $\omega \not\in \Gamma_{ij}$ we get that $\omega \not\in \Gamma$. Of course $\omega \not\in \alpha$ because $\alpha$ is an atom.

(iii-bis $\Rightarrow$ iii) Consider $t \in \text{nf}_\beta \mathcal{T}(M)$ such that $\Gamma \vdash t : \sigma$ where $\Gamma$ and $\sigma$ satisfy the hypotheses of (iii-bis). We proceed by induction on the structure of the $\beta$-normal $t$.

**Case** $t = \lambda x.t'$ where $t'$ is $\beta$-normal. By applying Theorem 3.7(ii) we have that $\Gamma, x : \mu \vdash t' : \tau$ holds for $\mu \in I_\Delta$ and $\tau \in \Gamma_\Delta$ such that $\sigma \simeq \mu \rightarrow \tau$. Since $\mathcal{D}$ preserves $\omega$-polarities, $\omega \not\in \mu$ entails $\omega \not\in \mu \rightarrow \tau$. As neither $\Gamma$ nor $\mu$ has negative occurrences of $\omega$, we have $\omega \not\in \mu \Gamma, x : \mu$ and $\omega \not\in \mu \tau$, so, by the induction hypothesis, we get that $t'$ does not have occurrences of $\mu$. Therefore $\mu$ does not occur in $\lambda x. t'$ either.

**Case** $t = y b_1 \cdots b_k$, for some $k \geq 0$, and each $b_i = [s_{i,1}, \ldots, s_{i,n_i}]$ (for $n_i \geq 0$) only contains $\beta$-normal terms. If $k = 0$ we are done, as $y$ does not contain $1$. Consider then the case $k > 0$. By iterating Theorem 3.7(iii') we know that there is a decomposition $\Gamma = \Gamma_0 \land (\land_{i=1}^{n_i} \land_{j=1}^{n_i} \Gamma_{ij})$ such that (setting $\mu_i = \land_{j=1}^{n_i} \tau_{ij}$):

$$\Gamma_0 \vdash y : \mu_1 \cdots \mu_k \rightarrow \sigma \quad \Gamma_{ij} \vdash s_{ij} : \tau_{ij} \quad \text{for } i = 1, \ldots, k \quad j = 1, \ldots, n_i$$

$$\Gamma \vdash y b_1 \cdots b_k : \sigma$$

By Theorem 3.7(i), we get that $\Gamma_0 = y : \tau$ for some $\tau \simeq \mu_1 \cdots \mu_k \rightarrow \sigma$. From this, it follows that $\Gamma(y) = \tau \land \mu$ for some $\mu$, so $\omega \not\in \Gamma$ entails that $\omega \not\in \Gamma$ and, as $\mathcal{D}$ preserves $\omega$-polarities, we get that $\omega \not\in \mu \rightarrow \mu \rightarrow \mu \rightarrow \sigma$. From this, on the one side we get that each $\mu_i$ is different from $\omega$ (that is, $n_i > 0$, so $b_i \neq 1$) and on the other side that $\omega \not\in \tau_{ij}$ holds for $1 \leq i \leq k$ and $1 \leq j \leq n_i$. We can therefore
apply the induction hypothesis to each derivation $\Gamma_{ij} \vdash s_{ij} : \tau_{ij}$ and conclude that the terms $s_{ij}$ do not contain 1, so neither does the term $yb_1 \cdots b_k$.

$(\text{iii-bis} \iff \text{iv})$ Let us suppose (iv). By Theorem 3.10, we have that $\Gamma \vdash M : \sigma$ holds if and only if there exists $s \in T(M)$ such that $\Gamma \vdash s : \sigma$. By Theorem 2.1 (strong normalization of $\Lambda'$) and Theorem 3.8(i) (subject reduction), the latter is equivalent to the existence of $t \in \text{nf}_\beta T(M)$ such that $\Gamma \vdash t : \sigma$. Therefore, (iv) is equivalent to (iii-bis).

For proving Lemma 4.4, we need the following remark and technical lemma.

**Remark A.2** In the model $D_\ast$, we have that $\sigma \simeq \star$ holds if and only if $\sigma$ is generated by the following grammar:

$$\gamma ::= \star \mid \gamma \rightarrow \gamma$$

In particular, $\mu \rightarrow \sigma \simeq \star$ entails that $\mu = \tau$ for some $\tau \simeq \star$ and $\sigma \simeq \star$.

**Lemma A.3** Let $N \in \Lambda$ be a $\beta$-normal form. If $\Gamma \vdash N : \sigma$, for some $\Gamma$ and $\sigma$ such that $\Gamma(x) \simeq \star$ for all $x \in \text{dom}(\Gamma)$ and $\sigma \simeq \star$, then $N$ is linear and $\text{dom}(\Gamma) = \text{fv}(N)$.

**Proof** We proceed by structural induction on $N$.

**Case** $N = \lambda x. N'$ where $N'$ is $\beta$-normal. From $\Gamma \vdash \lambda x. N' : \sigma$ we get, by Theorem 3.7(ii), that $\Gamma, x : \mu \vdash N' : \tau$ for some $\mu, \tau$ such that $\mu \rightarrow \tau \simeq \sigma$ and, by transitivity of $\simeq$, we get that $\mu \rightarrow \tau \simeq \star$ holds. By Remark A.2 this entails $\mu = \gamma$ for some $\gamma \simeq \star$ and $\tau \simeq \star$, therefore we can apply the induction hypothesis and get that $N'$ is linear and $\text{dom}(\Gamma, x : \gamma) = \text{fv}(N')$. Thus, $\lambda x. N'$ is also linear and $\text{dom}(\Gamma) = \text{fv}(N') - \{x\} = \text{fv}(\lambda x. N')$ which is what we are meant to prove.

**Case** $N = y_1 N_1 \cdots N_k$ such that $N_1, \ldots, N_k$ are $\beta$-normal. By Theorem 3.7(iii) and there is a decomposition $\Gamma = \Gamma_0 \wedge (\wedge_{i=1}^k \wedge_{j=1}^{n_i} \Gamma_{ij})$ such that $\Gamma_0 \vdash y : \mu_1 \rightarrow \cdots \rightarrow \mu_k \rightarrow \sigma$ holds for some $\mu_i = \tau_{i1} \wedge \cdots \wedge \tau_{im}$ and $\Gamma_{ij} \vdash N_i : \tau_{ij}$ is derivable for all $1 \leq i \leq k$ and $1 \leq j \leq n_i$. By Theorem 3.7(i), $\Gamma_0 = y : \gamma$, for a type $\gamma \simeq \mu_1 \rightarrow \cdots \rightarrow \mu_k \rightarrow \sigma$. As $\Gamma_0(y) = \Gamma(y) = \gamma \simeq \star$ we also have by transitivity of $\simeq$ that $\mu_1 \rightarrow \cdots \rightarrow \mu_k \rightarrow \sigma \simeq \star$ which entails by Remark A.2 that $\mu_i = \tau_i$ (i.e. $n_i = 1$) and $\tau_i \simeq \star$. Therefore we have $\Gamma = \Gamma_0 \wedge (\bigwedge_{i=1}^k \Gamma_i)$ and $\Gamma_i \vdash N_i : \tau_i$ for some $\Gamma_i$ such that $\Gamma_i(x) \simeq \star$ for all $x \in \text{dom}(\Gamma_i)$ and $\tau_i \simeq \star$.

By the induction hypothesis we get that each $N_i$ is linear and $\text{dom}(\Gamma_i) = \text{fv}(N_i)$. We conclude that $y_1 N_1 \cdots N_k$ is linear and $\text{dom}(\Gamma) = \text{dom}(\Gamma_0) \cup (\bigcup_{i=1}^k \text{dom}(\Gamma_i)) = \text{fv}(y_1 N_1 \cdots N_k)$.

**Lemma 4.4** Let $M \in \Lambda$ and $\Gamma = x_1 : \star, \ldots, x_n : \star$. Then $\Gamma \vdash^{D_\ast} M : \star$ if and only if $M$ has a linear $\beta$-normal form and $\text{dom}(\Gamma) = \text{fv}(\text{nf}_\beta(M))$.

**Proof** ($\Rightarrow$) By Theorem 4.2, the $\text{rgm} D_\ast$ preserves $\omega$-polarities. As $\omega$ does not occur positively nor negatively in $\star$, we can deduce by Lemma 4.3 that $M$ has a $\beta$-normal form. By subject reduction, we derive $\Gamma \vdash \text{nf}_\beta(M) : \star$ and, by Lemma A.3, we conclude that $\text{nf}_\beta(M)$ is linear.

($\Leftarrow$) Suppose that $M \in \Lambda$ has a linear $\beta$-normal form and that the environment $\Gamma = x_1 : \star, \ldots, x_n : \star$ is such that $\text{dom}(\Gamma) = \text{fv}(\text{nf}_\beta(M))$. It is enough to
prove that $\Gamma \vdash \text{nfl}_\beta(M) : \star$ is derivable, then one concludes by subject expansion (Theorem 3.8(i)) that $\Gamma \vdash M : \star$ holds. We proceed by induction on $\text{nfl}_\beta(M)$.

**Case** $\text{nfl}_\beta(M) = \lambda x.N'$ where $N'$ is $\beta$-normal. Obviously, $N'$ is linear and $\text{dom}(\Gamma, x : \star) = \text{fv}(N')$, so by the induction hypothesis

$$
\Gamma, x : \star \vdash N': \star \quad (1\text{am})
$$

is also derivable.

**Case** $\text{nfl}_\beta(M) = y N_1 \cdots N_k$ such that $N_1, \ldots, N_k$ are $\beta$-normal. We let $\Gamma_i$ to be the environment such that $\Gamma_i(x) = \emptyset$ if $x \in \text{fv}(N_i)$ and $\Gamma_i(x) = \omega$ otherwise. As the $N_i$'s are linear, we derive $\Gamma_i \vdash N_i : \star$ by the induction hypothesis. Then we can derive (for $\Gamma_0 = y : \star$):

$$
\frac{\frac{\frac{\frac{\Gamma_0 \vdash y : \star}{\star \leadsto \cdots \leadsto \star \leadsto \star}}{\Gamma_0 \vdash y : \star \leadsto \cdots \leadsto \star \leadsto \star \leadsto \star}}{\Gamma_0 \vdash \lambda x. N_1 \cdots N_k : \star \quad (1\text{am})}}{\Gamma \vdash \lambda x. N_1 \cdots N_k : \star}
$$

To conclude, it is enough to check that $\Gamma = \Gamma_0 \land (\land_{i=1}^k \Gamma_i)$. □

### A.4 Omitted proofs of Section 5

**Lemma A.4** Let $M \in \Lambda$ be a $\beta$-normal form such that $M \rightarrow_{\eta} x$. For all $a \in M^*$, we have that either $E(a, M^*) = \perp^{x}_{\eta(x)}$ or $E(a, M^*) \rightarrow_{\eta^*} x$.

**Proof** Since $M$ is $\beta$-normal, it has the shape $\lambda x_1 \ldots x_n.x N_1 \cdots N_k$. As $M \rightarrow_{\eta} x$ we get that $n = m$, $x \neq x_i$ and $N_i \rightarrow_{\beta} x_i$ for all $i \in \{1, \ldots, n\}$.

We proceed by induction on $a$.

**Case** $a = \perp$. Then $E(a, M^*) = \perp^{x}_{\eta(x)}$ by Definition 5.8.

**Case** $a = \lambda x_1 \ldots x_n.x a_1 \cdots a_n$ with $a_i \in N_i^*$ for all $i \in \{1, \ldots, n\}$. By induction hypothesis, either $E(a_i, N_i^*) \rightarrow_{\eta^*} x_i$ or $E(a_i, N_i^*) \rightarrow_{\eta^*} \perp_{\eta(x_i)}$ so $E(a, M^*) = \lambda x_1 \ldots x_n x E(a_1, N_1^*) \cdots E(a_n, N_n^*) \rightarrow_{\eta^*} x$. □

**Lemma 5.11** Let $T = \lambda \vec{x}. y.z T_1 \cdots T_{k+1}$ be a Böhm tree such that $T_{k+1}$ is finite, $T_{k+1} \rightarrow_{\eta} y$ and $y \notin \text{fv}(z T_1 \cdots T_k)$. Then $E(T^*) \rightarrow_{\eta^*} E((\lambda \vec{x}. z T_1^* \cdots T_k^*) \downarrow)$.

**Proof** We first prove that, given $a \in T^*$, there exists $a' \in (\lambda \vec{x}. z T_1^* \cdots T_k^*) \downarrow$ such that $E(a, T^*) \rightarrow_{\eta^*} E(a', (\lambda \vec{x}. z T_1^* \cdots T_k^*) \downarrow)$. We split into cases depending on $a$.

**Case** $a = \perp$. Then $E(a, T^*) = E(\perp, T^*) = E(\perp, (\lambda \vec{x}. z T_1^* \cdots T_k^*) \downarrow)$. From the fact that $T_{k+1}$ is finite, we get that $T_{k+1} \in \mathcal{N}$ and since $T_{k+1} \rightarrow_{\eta} y$ we have that $T_{k+1}$ is $\perp$-free. As $y \notin \text{fv}(z T_1 \cdots T_k)$, there is a $\perp$-free $c_1 \in T^*$ such that $c_1 \rightarrow_{\eta} z$ if and only if there exists a $\perp$-free $c_2 \in (\lambda \vec{x}. z T_1^* \cdots T_k^*) \downarrow$ such that $c_2 \rightarrow_{\eta} z$. Therefore $E(\perp, (\lambda \vec{x}. z T_1^* \cdots T_k^*) \downarrow) = E(\perp, (\lambda \vec{x}. z T_1^* \cdots T_k^*) \downarrow)$, so $a' = \perp$.

**Case** $a = \lambda \vec{x}. z a_1 \cdots a_{k+1}$ with $a_i \in T_i^*$ for $1 \leq i \leq k+1$. By definition, we have $E(a, T^*) = \lambda \vec{x}. y.E(a_1, T_1^*) \cdots E(a_k, T_k^*) E(a_{k+1}, T_{k+1}^*)$. By hypothe-
sis, $T_{k+1}$ is actually a $\lambda$-term (i.e., finite and $\perp$-free) such that $T_{k+1} \eta y$ so, by Lemma 5.11, either $E(a_{k+1}, T_{k+1}) \eta y$ or $E(a_{k+1}, T_{k+1}) \eta y$. By Remark 5.9 $y \notin \text{fv}(zT_1 \cdots T_k)$ entails $y \notin \text{fv}(zE(a_1, T_1') \cdots E(a_k, T_k'))$, hence in both cases we get $E(a, T^*) \eta y \lambda \vec{x}. zE(a_1, T_1') \cdots E(a_k, T_k') \in E((\lambda \vec{x}. zT_1^* \cdots T_k^*) \downarrow)$. Therefore the $a'$ we were looking for is just $\lambda \vec{x}. z a_1 \cdots a_k$.

Second, we prove that for every $a' \in (\lambda \vec{x}. z T_1^* \cdots T_k^*) \downarrow$ there is $a \in T^*$ such that $E(a, T^*) \eta y E(a', (\lambda \vec{x}. z T_1^* \cdots T_k^*) \downarrow)$. Again, we split into cases depending on $a'$.

Case $a' = \perp$. It is enough to take $a' = \perp$ and reason as above.

Case $a' = \lambda \vec{x}. z a_1' \cdots a_k'$ with $a_i' \in T_i^*$ for all $1 \leq i \leq k$. Clearly, $\perp \in T_{k+1}^*$ and $E(\perp, T_{k+1}^*) = \downarrow y$, since by hypothesis $T_{k+1}$ is finite and $T_{k+1} \eta y$. Therefore, for $a = \lambda \vec{x}. z a_1' \cdots a_k' \perp \in T^*$ we have

$$E(a, T^*) = \lambda \vec{x}. z \lambda \vec{x}. z(a_1', T_1^*) \cdots \lambda \vec{x}. z(a_k', T_k^*) \eta y$$

$$E(a, T^*) = \lambda \vec{x}. z \lambda \vec{x}. z(a_1', T_1^*) \cdots \lambda \vec{x}. z(a_k', T_k^*) \eta y$$

$$E(a', T^*) = \lambda \vec{x}. z T_1^* \cdots T_k^* + 1.$$  

We conclude as $E(a, T^*) \in E(T^*)$. 

Lemma A.5 For all Böhm trees $T$, we have $\eta(T)^* = \nf_{\eta^*}(E(T^*))^\gamma$.

Proof We proceed by co-induction on $T$.

If $T = \perp$, then $\eta(T)^* = \{ \perp \} = \{ \perp^0 \} = \{ \gamma \eta(\perp, \perp) \} = \gamma(\nf_{\eta^*}(E(T^*))^\gamma$.

Otherwise, the Böhm tree $T$ can be written in a unique way as $T = \lambda x_1 \cdots x_n y_1 \cdots y_m \cdot z T_1 \cdots T_k T_1' \cdots T_m'$ (for some $n, m, k \geq 0$) such that:

- $y_i \notin \text{fv}(z T_1 \cdots T_k)$, $T_i$ finite and $T_i \eta y_i$ for all $i \in \{1, \ldots, m\}$,
- $x_n \in \text{fv}(z T_1 \cdots T_k)$ or $T_k$ is infinite, or $T_k$ is finite but does not $\eta$-reduce to $x_n$. 


The following equalities hold:

\[ \eta(T)^* = \lambda \vec{x}. \eta(T_1)^* \cdots \eta(T_k)^* \cup \{ \bot \} \]

by def. of \( \eta(-) \)

\[ = \lambda \vec{x}. \neg nf_{\eta'}(\epsilon(T_1^*))^\vdash \cdots \neg nf_{\eta'}(\epsilon(T_k^*))^\vdash \cup \{ \bot \} \]

by co-IH

\[ = \neg \lambda \vec{x}. \neg nf_{\eta'}(\epsilon(T_1^*)) \cdots \neg nf_{\eta'}(\epsilon(T_k^*))^\vdash \]

\[ \cup \{ \epsilon(\bot, (\lambda \vec{x}. zT_1^* \cdots T_k^*) \downarrow) \}^\vdash \]

by def. of \( \neg \cdot \vdash \)

\[ = \neg \lambda \vec{x}. \neg nf_{\eta'}(\epsilon(T_1^*)) \cdots \neg nf_{\eta'}(\epsilon(T_k^*))^\vdash \]

\[ \cup \{ nf_{\eta'}(\epsilon(\bot, (\lambda \vec{x}. zT_1^* \cdots T_k^*) \downarrow)) \}^\vdash \]

by def. of \( nf_{\eta'}(-) \)

\[ = \neg nf_{\eta'}(\lambda \vec{x}. zT_1^* \cdots T_k^*) \downarrow \]

by def. of \( \epsilon(-) \)

\[ = \neg nf_{\eta'}(\epsilon(T^*))^\vdash \]

by Lemma 5.11. \( \square \)

5.12 For all \( M \in \Lambda \), we have \( BT^\eta(M)^* = \neg nf_{\eta'}(\epsilon(BT^\eta(M))^*)^\vdash \).

**Proof** Since \( BT^\eta(M) = \eta(BT^\eta(M)) \), the result follows directly by Lemma A.5. \( \square \)

**Lemma A.6** Let \( M \in \Lambda \) be a \( \beta \)-normal form such that \( M \rightarrow^\eta x \). Then for all \( t \in T(M) \), we have \( \mathcal{L}(t, T(M)) \rightarrow_{nf'} x \).

**Proof** By hypothesis, \( M \) has the shape \( \lambda x_1 \ldots x_n.xM_1 \cdots M_n \) (for some \( n \geq 0 \)) such that, for all \( i \in \{1, \ldots, n\} \), \( x \neq x_i \) and \( M_i \) is a \( \beta \)-normal form such that \( M_i \rightarrow^\eta x_i \). We proceed by induction on \( t \). Since \( t \in T(M) \), we have \( t = \lambda x_1 \ldots x_n.xb_1 \cdots b_n \) such that \( b_i \in M_t(T(M_i)) \) for every \( 1 \leq i \leq n \). If \( n = 0 \) we are done. Otherwise, by Definition 5.1 we have \( \mathcal{L}(t, T(M)) = \lambda x_1 \ldots x_n.xL(b_1, M_t(T(M_1))) \cdots L(b_n, M_t(T(M_n))). \)

Suppose \( b_n = [t_1, \ldots, t_k] \) with \( t_j \in T(M_n) \) for all \( j \in \{1, \ldots, k\} \).

If \( k = 0 \) then, by Definition 5.1, \( L(b_n, M_t(T(M_n))) = 1^{x_n}_{\eta(x_n)} \) because \( M_n \rightarrow^\eta x_n \) entails that there is \( s \in \bigcup M_t(T(M_n)) = T(M_n) \) such that \( s \rightarrow_{nf'} x_n \). Therefore:

\[ \mathcal{L}(t, T(M_n)) \rightarrow_{nf'} \lambda x_1 \ldots x_n.xL(b_1, M_t(T(M_1))) \cdots L(b_n-1, M_t(T(M_n-1)))1^{x_n}_{\eta(x_n)} \]

If \( k > 0 \), then by induction hypothesis \( \mathcal{L}(t_{njq}, T(M_n)) \rightarrow_{nf'} x_n \). Therefore,

\[ \mathcal{L}(t, T(M_n)) \rightarrow_{nf'} \lambda x_1 \ldots x_{n-1}.xL(b_1, M_t(T(M_1))) \cdots L(b_{n-1}, M_t(T(M_n-1)))[x_n^k] \]

By iterating this reasoning on \( b_1, \ldots, b_{n-1} \) we conclude that \( \mathcal{L}(t, T(M)) \rightarrow_{nf'} x. \) \( \square \)
\textbf{5.13} Let $T = \lambda \vec{x}.z T_1 \cdots T_{k+1}$ be a Böhm tree such that $T_{k+1}$ is finite, $T_{k+1} \rightarrow_n y$ and $y \notin \text{fv}(z T_1 \cdots T_k)$. Then $\mathcal{L}(T(T)) \Rightarrow_{n'} \mathcal{L}(T(\lambda \vec{x}.z T_1 \cdots T_k))$.

\textbf{Proof} We first take $t \in \mathcal{T}(T)$, that is $t = \lambda \vec{x}.y.z b_1 \cdots b_{k+1}$ with $b_i \in \mathcal{M}_t(\mathcal{T}(T_i))$, and show that $\mathcal{L}(t, T(T)) \Rightarrow_{n'} \mathcal{L}(t', T(\lambda \vec{x}.z T_1 \cdots T_k))$ holds for $t' = \lambda \vec{x}.z b_1 \cdots b_k \in \mathcal{L}(T(\lambda \vec{x}.z T_1 \cdots T_k))$. By definition of the labeling $\mathcal{L}(-)$, we have $\mathcal{L}(t, T(T)) = \lambda \vec{x}.y.z \mathcal{L}(b_1, \mathcal{M}_t(\mathcal{T}(T_1))) \cdots \mathcal{L}(b_{k+1}, \mathcal{M}_t(\mathcal{T}(T_{k+1}))))$. By Remark 5.4 we have that $y \notin \text{fv}(z T_1 \cdots T_k)$ implies $y \notin \text{fv}(z \mathcal{L}(b_1, \mathcal{M}_t(\mathcal{T}(T_1))) \cdots \mathcal{L}(b_k, \mathcal{M}_t(\mathcal{T}(T_k))))$.

Suppose that $b_{k+1} = [t_1, \ldots, t_n]$, we split into cases depending on $n$.

\textbf{Case $n = 0$.} As $T_{k+1} \rightarrow_n y$, then $T_{k+1}$ is $\perp$-free finite tree, and therefore there exists an $s \in \mathcal{T}(T_{k+1})$ without empty bags such that $s \rightarrow_n y$. Hence $\mathcal{L}(b_{k+1}, \mathcal{M}_t(\mathcal{T}(T_{k+1}))) = \mathcal{L}(1, \mathcal{M}_t(\mathcal{T}(T_{k+1}))) = 1^y_{\eta(y)}$ since $s \in \bigcup \mathcal{M}_t(\mathcal{T}(T_{k+1})) = \mathcal{T}(T_{k+1})$. Therefore we have:

$$
\mathcal{L}(t, T(T)) = \lambda \vec{x}.y.z \mathcal{L}(b_1, \mathcal{M}_t(\mathcal{T}(T_1))) \cdots \mathcal{L}(b_{k+1}, \mathcal{M}_t(\mathcal{T}(T_{k+1})))) = \lambda \vec{x}.y.z \mathcal{L}(b_1, \mathcal{M}_t(\mathcal{T}(T_1))) \cdots \mathcal{L}(b_k, \mathcal{M}_t(\mathcal{T}(T_k))))1^y_{\eta(y)} 
$$

As $T_{k+1} \rightarrow_n y$, then $T_{k+1}$ is $\perp$-free finite tree, and therefore there exists an $s \in \mathcal{T}(T_{k+1})$ without empty bags such that $s \rightarrow_n y$. Thus $\mathcal{L}(1, \mathcal{M}_t(\mathcal{T}(T_{k+1}))) = 1^y_{\eta(y)}$ since $s \in \bigcup \mathcal{M}_t(\mathcal{T}(T_{k+1})) = \mathcal{T}(T_{k+1})$. Hence, we have:

$$
\mathcal{L}(t, T(T)) = \lambda \vec{x}.y.z \mathcal{L}(b_1, \mathcal{M}_t(\mathcal{T}(T_1))) \cdots \mathcal{L}(b_k, \mathcal{M}_t(\mathcal{T}(T_k)))) \mathcal{L}(1, \mathcal{M}_t(\mathcal{T}(T_{k+1})))) = \lambda \vec{x}.y.z \mathcal{L}(b_1, \mathcal{M}_t(\mathcal{T}(T_1))) \cdots \mathcal{L}(b_k, \mathcal{M}_t(\mathcal{T}(T_k))))1^y_{\eta(y)} 
$$

Second, we take $s \in \mathcal{T}(\lambda \vec{x}.z T_1 \cdots T_k)$, i.e. $s = \lambda \vec{x}.z b_1 \cdots b_k$ with $b_i \in \mathcal{M}_t(\mathcal{T}(T_i))$, and show that $\mathcal{L}(t, T(T)) \rightarrow_{n'} \mathcal{L}(s, T(\lambda \vec{x}.z T_1 \cdots T_k))$ for $t = \lambda \vec{x}.y.z b_1 \cdots b_{k+1} \in T(T)$. As $T_{k+1} \rightarrow_n y$, then $T_{k+1}$ is $\perp$-free finite tree, and therefore there exists an $s \in \mathcal{T}(T_{k+1})$ without empty bags such that $s \rightarrow_{n'} y$. Thus $\mathcal{L}(1, \mathcal{M}_t(\mathcal{T}(T_{k+1}))) = 1^y_{\eta(y)}$ since $s \in \bigcup \mathcal{M}_t(\mathcal{T}(T_{k+1})) = \mathcal{T}(T_{k+1})$. Hence, we have:

$$
\mathcal{L}(t, T(T)) = \lambda \vec{x}.y.z \mathcal{L}(b_1, \mathcal{M}_t(\mathcal{T}(T_1))) \cdots \mathcal{L}(b_k, \mathcal{M}_t(\mathcal{T}(T_k)))) \mathcal{L}(1, \mathcal{M}_t(\mathcal{T}(T_{k+1})))) = \lambda \vec{x}.y.z \mathcal{L}(b_1, \mathcal{M}_t(\mathcal{T}(T_1))) \cdots \mathcal{L}(b_k, \mathcal{M}_t(\mathcal{T}(T_k))))1^y_{\eta(y)} 
$$

This completes the proof. \hfill \Box
Lemma A.7 For all Böhm tree $T$ the following equality holds:

$$\mathcal{T}(\text{nf}_{\eta'} \mathcal{E}(T^*)^\gamma) = \text{nf}_{\eta'} \mathcal{L}(\mathcal{T}(T))^\gamma$$

Proof We proceed by co-induction on $T$.

If $T = \bot$, then the equality follows because $\mathcal{T}(\bot) = \emptyset$.

Otherwise, the Böhm tree $T$ can be written in a unique way as $T = \lambda x_1 \ldots x_n y_1 \ldots y_m z T_1 \ldots T_k T'_1 \ldots T'_m$ (for some $n, m, k \geq 0$) such that:

- $y_i \notin \text{fv}(z T_1 \ldots T_k)$, $T'_i$ is finite and $T'_i \xrightarrow{\eta} y_i$ for all $i \in \{1, \ldots, m\}$,
- $x_n \in \text{fv}(z T_1 \ldots T_k)$ or $T_k$ is infinite, or $T_k$ is finite but does not $\eta$-reduce to $x_n$.

Therefore, the following equalities hold:

$$\mathcal{T}(\text{nf}_{\eta'} \mathcal{E}(T^*)^\gamma) = \mathcal{T}(\text{nf}_{\eta'} \mathcal{E}((\lambda x_1 \ldots x_n z T_1 \ldots T_k) \downarrow)^\gamma)$$ by Lemma 5.11

$$= \mathcal{T}(\text{nf}_{\eta'} \mathcal{E}(\mathcal{T}(T_1)^*)^\gamma) \cup \{ \mathcal{E}(\bot, \lambda \vec{x}. z T_1^* \cdots T_k^*) \}$$ by def. of $\mathcal{E}(\cdot)$

$$= \mathcal{T}(\text{nf}_{\eta'} \mathcal{E}(\mathcal{T}(T_1)^*)^\gamma) \cup \{ \mathcal{E}(\bot, \lambda \vec{x}. z T_1^* \cdots T_k^*) \}$$ by def. of $\mathcal{T}(\cdot)$

$$= \lambda \vec{x}. z \mathcal{M}_t(\mathcal{T}(\text{nf}_{\eta'} \mathcal{E}(\mathcal{T}(T_1)^*)^\gamma)) \cdots \mathcal{M}_t(\mathcal{T}(\text{nf}_{\eta'} \mathcal{E}(\mathcal{T}(T_k)^*)^\gamma))$$ since $\mathcal{T}(\bot) = \emptyset$

$$= \lambda \vec{x}. z \mathcal{M}_t(\text{nf}_{\eta'} (\mathcal{L}(\mathcal{T}(T_1))^\gamma)) \cdots \mathcal{M}_t(\text{nf}_{\eta'} (\mathcal{L}(\mathcal{T}(T_k))^\gamma))$$ by co-IH

$$= \text{nf}_{\eta'} (\lambda \vec{x}. z \mathcal{M}_t(\mathcal{T}(T_1)) \cdots \mathcal{M}_t(\mathcal{T}(T_k))^\gamma)$$ by def. of $\text{nf}_{\eta'}$

$$= \text{nf}_{\eta'} \mathcal{L}(\lambda \vec{x}. z \mathcal{M}_t(\mathcal{T}(T_1)) \cdots \mathcal{M}_t(\mathcal{T}(T_k))^\gamma)$$ by def. of $\mathcal{L}(\cdot)$

$$= \text{nf}_{\eta'} \mathcal{L}(\mathcal{T}(\lambda \vec{x}. z T_1 \cdots T_k))^\gamma$$ by def. of $\mathcal{T}(\cdot)$

$$= \text{nf}_{\eta'} \mathcal{L}(\mathcal{T}(T))^\gamma$$ by Lemma 5.13.

**5.14** For all $M \in \Lambda$, $\mathcal{T}(\text{nf}_{\eta'} \mathcal{E}(\mathcal{B}(M)^*)^\gamma) = \text{nf}_{\eta'} \mathcal{L}(\mathcal{T}(\mathcal{B}(M))^\gamma)$.

Proof It follows directly from Lemma A.7. □

272
The Coinductive Resumption Monad

Maciej Piróg and Jeremy Gibbons

Department of Computer Science
University of Oxford

Abstract
Resumptions appear in many forms as a convenient abstraction, such as in semantics of concurrency and as a programming pattern. In this paper we introduce generalised resumptions in a category-theoretic, coalgebraic context and show their basic properties: they form a monad, they come equipped with a corecursion scheme in the sense of Adámek et al.’s notion of completely iterative monads (cims), and they enjoy a certain universal property, which specialises to the coproduct with a free cim in the category of cims.

Keywords: resumptions, completely iterative monads, coalgebra

1 Introduction

1.1 Resumptions

Resumptions were introduced by Milner [32] to denote the external behaviour of a communicating agent in concurrency theory. In categorical terms, as given by Abramsky [1], a resumption is an element of the carrier of the final coalgebra \( \nu R \) of the functor \( RX = (X \times O)^I \) on \( \text{Set} \), where \( I \) and \( O \) are the sets of possible inputs and outputs respectively. Informally, a resumption is a function that consumes some input and returns some output paired with another resumption, and finality implies that the process of consuming and producing can be iterated indefinitely.

There are a lot of possible generalisations, for example to the final coalgebra of the functor \( P_{\text{fin}}(((-) \times O)^I) \) for agents with finitely-branching nondeterministic behaviour, or, depending on the computational effect realised by the agent, any monad in place of \( P_{\text{fin}} \), as studied by Hasuo and Jacobs [24].

The idea of ‘resumable’ computations appeared also in the context of computational effects and monadic programming. Cenciarelli and Moggi [13] defined what

1 maciej.pirog@cs.ox.ac.uk
2 jeremy.gibbons@cs.ox.ac.uk
they called the generalised resumption monad transformer as \( TA = \mu X. M(\Sigma X + A) \),
where \( M \) is a monad, \( \Sigma \) is an endofunctor, and \( A \) is an object of variables, which allows to sequentially compose resumptions. The resumption monad can be alternatively given by \( M(\Sigma M)^* \), a composition of \( M \) and the free monad generated by the composition of \( \Sigma \) and \( M \). It is canonical in the sense that it is the coproduct of \( M \) and \( \Sigma^* \) in the category of monads and monad morphisms, as shown by Hyland, Plotkin, and Power [26].

The resumption monad is given by an initial algebra, so it is not exactly a generalisation of the resumptions studied by Milner and others. Intuitively, it models resumptions that can be iterated only finitely many times. Thus, it is natural to ask about final coalgebras of functors of the shape \( M(\Sigma(-) + A) \). Indeed, Goncharov and Schröder [21] used monads of the shape \( TA = \nu X. M(\Sigma X + A) \) to give semantics to concurrent processes with generic effects, while the monad \( TA = \nu X. M(\Sigma X + A) \) was discussed by the present authors [39] under the name “coinductive resumption monad”. In this paper, we further generalise the latter construction and take a closer look at its properties.

1.2 Coinduction

Usually, an effect-free data structure is called coinductive if it is given by the carrier of a final coalgebra. Informally, finality means that a coalgebra \( c : X \to FX \) describes one step, which, repeated indefinitely, builds a structure of the type \( \nu F \) layer by layer. In the monadic world, however, the steps are often meant to interact – if we imagine that monadic values are computations, all the steps should be composed (monadically speaking, multiplied out) into one, big computation; if we view monads as algebraic theories, we should take into account that operations in one layer have their arguments in the next layer. Obviously, not every monad is coinductive in this sense, because the notion of multiplication of infinitely many layers is not always well-defined. Thus, to capture the notion of coinduction in the monadic context, we adopt a property called complete iterativity, introduced by Elgot et al. [16] and later studied by Adámek et al. [4,30]. A monad \( M \) is completely iterative (is a ‘cim’) if it is equipped with an additional coinductive structure: certain (‘guarded’) morphisms \( e : X \to M(X + A) \), where \( X \) represents variables (seeds of the corecursion) and \( A \) is an object of parameters (final values), have unique solutions \( e^\dagger : X \to MA \) coherent with the monadic structure of \( M \).

Not too surprisingly, the usual notion of coinduction is captured by free completely iterative monads (informally: layers do not interact). The free completely iterative monad generated by an endofunctor \( F \) is given as \( F^\infty A = \nu X. FX + A \) with the monadic structure given by substitution in \( A \). An ordinary coalgebra \( X \to FX \) can be seen as a guarded morphism \( X \to F^\infty(X + 0) \), where 0 is the initial object of the base category, with the unique solution \( X \to F^\infty0 \cong \nu F \).
1.3 Contributions

The monad \( TA = \nu X.M(\Sigma X + A) \) can alternatively be given as \( M(\Sigma M)^\infty \). In Section 3, we generalise this construction to \( MS^\infty \), where \( S \) is any right module of \( M \) (all the necessary definitions and notations are given in Section 2). We give it a monadic structure and prove that it is completely iterative. Moreover, if \( M \) is also a cim, \( MS^\infty \) is a cim both ‘vertically’ (informally, we build new levels of the free structure) and ‘horizontally’ (we unfold more \( M \) structure) simultaneously.

In Section 4, we turn our attention back to the instance \( M(\Sigma M)^\infty \). We show that it enjoys a certain universal property, which entails that it is the coproduct of \( \Sigma^\infty \) and \( M \) in the category of cims. In Section 5, we discuss corollaries and potential applications of our construction in semantics and programming.

Because of space limitations we present only short outlines of some proofs. For the full proofs, consult the associated technical appendix available online at http://www.cs.ox.ac.uk/people/maciej.pirog/crm-appendix.pdf.

2 Idealised and completely iterative monads

In the rest of the paper, we work in a base category \( B \) with finite coproducts. For brevity, we assume strict associativity of the coproduct bifunctor. The left and right injections are called \( \text{inl} \) and \( \text{inr} \) respectively. For an endofunctor \( F : B \to B \), we denote the carrier of the initial \( F \)-algebra as \( \mu F \), and the carrier the final \( F \)-coalgebra as \( \nu F \). The unique morphism from a coalgebra \( \langle A, g : A \to FA \rangle \) to the final coalgebra \( \langle \nu F, \xi : \nu F \to F\nu F \rangle \) is written as \( \{ g \} \). We use the letters \( M,N,T \) for monads. Their monadic structure is always denoted as \( \eta \) (unit) and \( \mu \) (multiplication), possibly with superscripts to assign the natural transformations to the appropriate monad. The category of monads and monad morphism is denoted as \( \text{Mnd} \), while the category of Eilenberg-Moore algebras of a monad \( M \) is denoted as \( \text{Malg} \).

Definition 2.1 Let \( M \) be a monad. An endofunctor \( M \) together with a natural transformation (an action) \( \bar{\mu} : \bar{M}M \to \bar{M} \) is called a (right) \( M \)-module if \( \bar{\mu} : \bar{M} \eta = \text{id} : \bar{M} \to \bar{M} \) and \( \bar{\mu} \cdot \bar{\mu}M = \bar{\mu} \cdot \bar{M} \mu : \bar{MM}^2 \to \bar{M} \).

We define a morphism between two \( M \)-modules \( \langle \bar{M}, \bar{\mu} \rangle \) and \( \langle \bar{\bar{M}}, \bar{\bar{\mu}} \rangle \) as a natural transformation \( f : \bar{M} \to \bar{\bar{M}} \) such that \( \bar{\bar{\mu}} \cdot fM = f \cdot \bar{\mu} : \bar{MM} \to \bar{\bar{M}} \).

Slightly abusing notation, we often denote a module \( \langle \bar{M}, \bar{\mu} \rangle \) simply as \( \bar{M} \).

Example 2.2 The following are examples of right modules:

(i) For a monad \( M \), the pair \( \langle M, \mu^M \rangle \) is an \( M \)-module.
(ii) Let \( m : M \to T \) be a monad morphism. Then, the pair \( \langle T, \mu^T \cdot Tm \rangle \) is an \( M \)-module.
(iii) Let \( \langle \bar{M}, \bar{\mu}^M \rangle \) be an \( M \)-module and \( F \) be an endofunctor. Then, \( \langle F\bar{M}, F\bar{\mu}^M \rangle \) is an \( M \)-module.
(iv) With the definitions as above, let \( \lambda : TM \to MT \) be a distributive law between
monads. The pair \(\langle MT, (\overline{p^M} \ast \mu^T) \cdot M\lambda T \rangle\) is a module of the induced monad \(MT\).

(v) If \(\langle M, \mu^M \rangle\) and \(\langle \tilde{M}, \tilde{\mu}^M \rangle\) are two \(M\)-modules, the pair \(\langle \tilde{M} + \tilde{M}, \tilde{p}^M + \tilde{\mu}^M \rangle\) is also an \(M\)-module.

(vi) Let \(F\) be an endofunctor with a right adjoint \(U\). Then, \(F\) is an \(UF\)-module with the action given by \(\varepsilon F : FUF \to F\), where \(\varepsilon\) is the counit of the adjunction.

Definition 2.3 An idealised monad is a triple consisting of a monad \(M\), an \(M\)-module \(\langle M, \mu^M \rangle\), and a module homomorphism \(\sigma : \langle \overline{M}, \overline{p}^M \rangle \to \langle M, \mu^M \rangle\). We say that \(M\) is idealised with \(\langle \overline{M}, \overline{p}^M \rangle\). If \(M = \overline{M} + \text{Id}\), we say that \(M\) is ideal. For an endofunctor \(F\), a natural transformation \(k : F \to M\) is ideal if it factors through \(\sigma\).

If not stated otherwise, for an idealised monad \(M\), by \(\mu^M\) we always denote the action of the associated module \(\overline{M}\), and by \(\sigma^M\) the associated module homomorphism.

Example 2.4 Extending Example 2.2 (iv) and (v), it holds that:

(i) Let \(M\) be idealised with \(\overline{M}\) and \(\lambda : TM \to MT\) be a distributive law between monads. The induced monad \(MT\) is idealised with \(\overline{M}\). The associated module morphism is given by \(\sigma^M T : \overline{MT} \to MT\).

(ii) Let \(M\) be idealised with \(\overline{M}\) as well as with \(\tilde{M}\). Then, \(M\) is idealised with \(\overline{M} + \tilde{M}\). The associated module morphism is given by \([\sigma, \sigma'] : \overline{M} + \tilde{M} \to M\), where \(\sigma\) and \(\sigma'\) are the respective associated morphisms of the two modules.

Definition 2.5 Let \(M\) be a monad idealised with \(\overline{M}\). A morphism \(e : X \to M(X + A)\) is called a guarded equation morphism if it factors as follows:

\[
X \xrightarrow{e} \overline{M}(X + A) + A \xrightarrow{[\sigma_{X+A}, \eta_{X+A} \cdot \text{in}_{X,A}]} M(X + A)
\]

We call a morphism \(e^\dagger : X \to MA\) a solution of \(e\) if the following diagram commutes:

\[
\begin{array}{ccc}
X & \xrightarrow{e^\dagger} & MA \\
\downarrow{e} & & \uparrow{\mu_A} \\
M(X + A) & \xrightarrow{M[e^\dagger, \eta_A]} & M^2A
\end{array}
\]

An idealised monad \(M\) is completely iterative (is a ‘cim’) if every guarded equation morphism has a unique solution.

A monad morphism \(m : M \to T\), where \(T\) is idealised with \(\langle T, \overline{p}^T \rangle\), is said to preserve solutions if there exists a natural transformation \(\overline{m} : \overline{M} \to T\), such that \(m \cdot \sigma^M = \sigma^T \cdot \overline{m} : \overline{M} \to T\).

We denote the category of all cims and solution-preserving monad morphisms as \(\text{Cim}\).

Note that we use a different notion of morphisms between cims than Adámek et al. [5], whose morphisms – called idealised monad morphisms – preserve also the
structure of modules. The name ‘solution-preserving’ comes from the fact that for an equation morphism \( e \) and \( m \) as in the definition above, it holds that \( m_{X+A} \cdot e \) is guarded and that \((m_{X+A} \cdot e)^\dagger = m_A \cdot e^\dagger \) (see the proof of a theorem by Milius [30, Prop. 5.9]).

An important example of a cim is the free cim \( \Sigma^\infty \) generated by an endofunctor \( \Sigma \). Intuitively, it captures the monad of non-well-founded \( \Sigma \)-terms. Given an endofunctor \( \Sigma \) (a signature), if the final coalgebra \( \langle \nuX.\Sigma X + A, \xiA \rangle \) exists for all objects \( A \), then we define \( \Sigma^\infty A = \nuX.\Sigma X + A \). One can show that it is functorial in \( A \), with the obvious action on morphisms, and that it has a monadic structure given by substitution in \( A \), which we denote as \( \eta^\infty \) and \( \mu^\infty \). The monad \( \Sigma^\infty \) is ideal and completely iterative. We define a natural transformation \( \text{emb}_A : \Sigma \rightarrow \Sigma^\infty \) as:

\[
\text{emb}_A = \left( \Sigma A \xrightarrow{\Sigma \eta^\infty} \Sigma \Sigma^\infty A \xrightarrow{\text{inl}} \Sigma \Sigma^\infty A + A \xrightarrow{\xi^{-1}} \Sigma^\infty A \right)
\]

As discussed by Aczel et al. [4], \( \Sigma^\infty \) is the free cim generated by \( \Sigma \). Intuitively, this means that every ideal interpretation of \( \Sigma \) in a cim \( M \) extends in a unique way to an interpretation of the entire structure in \( M \). Formally:

**Theorem 2.6** For a cim \( M \) and an ideal natural transformation \( k : \Sigma \rightarrow M \), there exists a unique monad morphism \( \iota(k) : \Sigma^\infty \rightarrow M \) such that \( k = \iota(k) \cdot \text{emb} \).

Moreover, the morphism \( \iota(k) \) preserves solutions.

We also need the following cancellation property:

**Lemma 2.7** For a cim \( M \) and a solution-preserving monad morphism \( m : \Sigma^\infty \rightarrow M \), the composition \( m \cdot \text{emb} \) is an ideal natural transformation, and \( \iota(m \cdot \text{emb}) = m \).

### 3 Monadic structure and complete iterativity

Let \( \langle S, \overline{\rho}^S \rangle \) be a right \( M \)-module such that \( S^\infty \) exists. We give a monadic structure to \( MS^\infty \) via a distributive law [11]. This construction is an adaptation of Hyland, Plotkin, and Power’s proof [26] that the inductive resumptions \( M(\Sigma M)^* \) form a monad. We use the fact due to Adámek, Milius, and Velebil [7] that the category of complete Elgot algebras is strictly monadic over the base category \( \mathbb{B} \). Note that we cannot employ Uustalu’s construction on parametrised monads [41] (successfully used by Goncharov and Schröder [21] in the special case of \( MM^\infty \)), since \( MS^\infty \) is not in general given by the carrier of a final coalgebra, and also we make extensive use of the distributive law later in this paper. We start by recalling the definition of Elgot algebras [7].

**Definition 3.1** Let \( H \) be an endofunctor. For two objects \( A \) and \( X \), we call a morphism \( e : X \rightarrow HX + A \) a flat equation morphism. We call a morphism \( e^\dagger : X \rightarrow A \) a solution in an \( H \)-algebra \( a : HA \rightarrow A \) if \( e^\dagger = [a \cdot He^\dagger, \text{id}] \cdot e \).

Just like in the case of cims, we denote the solutions in Elgot algebras by \((-)^\dagger \) or \((-)^\ddagger \). This overloading should not impose any confusion, since we are always clear about the types.
Definition 3.2 For flat equation morphisms \( e : X \rightarrow HX + Y \) and \( f : Y \rightarrow HY + A \), and a morphism \( h : Y \rightarrow Z \), we define two operations:

\[
\begin{align*}
    h \circ e &= \left( X \xrightarrow{e} HX + Y \xrightarrow{id + h} HX + Z \right) \\
    f \circ e &= \left( X + Y \xrightarrow{[e, \text{inr}]} HX + Y \xrightarrow{id + f} HX + HY + A \xrightarrow{[Hf, H\text{inr}] + id} H(X + Y) + A \right)
\end{align*}
\]

Definition 3.3 For an endofunctor \( H \), a complete Elgot \( H \)-algebra is a triple \( \langle A, a : HA \rightarrow A, (-)\dagger \rangle \), where \((-)\dagger\) assigns to every flat equation morphism \( e : X \rightarrow HX + A \) a solution \( e\dagger : X \rightarrow A \) such that the following two conditions hold:

- (Functoriality) For two equation morphisms \( e : X \rightarrow HX + A \) and \( f : Y \rightarrow HY + A \) understood as \( H(-) + A \) coalgebras, let \( h : X \rightarrow Y \) be a coalgebra homomorphism, that is \( f \cdot h = (Hh + \text{id}_A) \cdot e \). Then, \( e\dagger = f\dagger \cdot h \).
- (Compositionality) For two equation morphisms \( e : X \rightarrow HX + Y \) and \( f : Y \rightarrow HY + A \) it holds that \( (f \circ e)\dagger = (f \circ e) \cdot \text{inl} \).

Definition 3.4 For two complete Elgot \( H \)-algebras \( \langle A, a, (-)\dagger \rangle \) and \( \langle B, b, (-)\dagger \rangle \), a morphism \( h : A \rightarrow B \) is said to preserve solutions if for every flat equation morphism \( e : X \rightarrow HX + A \) it holds that \( (h \circ e)\dagger = h \cdot e\dagger \). Complete Elgot \( H \)-algebras and solution-preserving morphisms form a category, which we denote as \( H\text{-ELGOT} \).

Theorem 3.5 (Adámek, Milius, Velebil [7]) The obvious forgetful functor \( U_{\text{Elg}} : H\text{-ELGOT} \rightarrow \mathbb{B} \) is strictly \( H^\infty \)-monadic (or simply ‘monadic’ in Mac Lane’s terminology [29, Ch. 6]), which entails \( H\text{-ELGOT} \cong H^\infty\text{-MALG} \).

Construction 3.6 Recall that \( \langle S, \Xi^S \rangle \) is a right \( M \)-module. For a complete Elgot algebra \( \langle A, a : SA \rightarrow A, (-)\dagger \rangle \) we define an algebra \( \langle MA, a' : SMA \rightarrow MA, (-)\dagger \rangle \) as follows:

\[
a' = \left( SMA \xrightarrow{\Xi^S} SA \xrightarrow{a} A \xrightarrow{\eta^M} MA \right)
\]

Let \( e : X \rightarrow SX + MA \) be a flat equation morphism. We define an auxiliary morphism \( |e| \) and a solution \( e\dagger \):

\[
|e| = \left( SX + A \xrightarrow{Se + \text{id}} S(SX + MA) + A \xrightarrow{\text{flat} + \text{id}} S(SX + A) + A \right)
\]

\[
e\dagger = \left( X \xrightarrow{e} SX + MA \xrightarrow{\text{inl} + \text{id}} SX + A + MA \xrightarrow{|e| + \text{id}} A + MA \xrightarrow{[\eta^M, \text{id}]} MA \right)
\]

where \( \text{flat}_{A,B} \) is given as:

\[
S(A + MB) \xrightarrow{S(\eta^M + \text{id})} S(MA + MB) \xrightarrow{S[\text{Mfin}, \text{Mfin}]} SM(A + B) \xrightarrow{\Xi^S} S(A + B)
\]

Lemma 3.7 The triple \( \langle MA, a', (-)\dagger \rangle \) from Construction 3.6 is a complete Elgot algebra. Moreover, the assignment \( \langle A, a, (-)\dagger \rangle \mapsto \langle MA, a', (-)\dagger \rangle \) on objects and \( f \mapsto \ldots \)
The composition $M S^\infty$ is a monad.

**Proof.** The assignment from Lemma 3.7 is a monad, so it is a lifting of $M$ to $S\text{-Elgot}$ with respect to $U_{\text{Elg}}$. Thus, by Theorem 3.5, it is a lifting of $M$ to $S^\infty\text{-MAlg}$. This induces a distributive law between monads $\lambda : S^\infty M \to MS^\infty$, which gives a monadic structure to $MS^\infty$. \qed

**Example 3.9** (a) Let $\mathbb{B}$ be Set, $D$ be the monad of discrete probability distributions, $O = \{a, b, \ldots\}$ be a set, and $\Sigma X = O \times X$ be an endofunctor. A value of the monad $D(\Sigma D)^\infty X$ is a countably branching, possibly infinite decision tree with elements of the set $O$ in nodes (except for the root), edges labelled with probabilities, and elements of $X$ in leaves. For illustration purposes, it is important that there is no label in the root – we take a random step before generating a label and a random step before reaching a leaf. This way, when we substitute a tree for a variable, we take two random steps before generating a label or reaching a leaf. The monadic structure of $D(\Sigma D)^\infty X$ takes care of flattening these to one random step by multiplying out the adjacent distributions; see Figure 1 for an example.

(b) In a cartesian closed category, we can define a version of the state monad that keeps track of all (possibly infinitely many) intermediate states. It is similar but not identical to ‘states’ given in [39]. Fix an object of states $A$ and consider the reader monad $RX = X^A$. By Example 2.2 (vi), the writer $WX = X \times A$ is an $R$-module. More directly, the action can be given as $\langle \text{ev}_X^A, \text{outr} \rangle : WRX = X^A \times A \to X \times A = WX$, where $\text{ev}$ is the evaluation morphism of the exponential object, $\text{outr}$ is the right projection, and $\langle \cdot, \cdot \rangle$ is the product mediator. Intuitively, for an initial state, the monad $RW^\infty = ((- \times X)^\infty)^A$ produces a (possibly infinite) stream of intermediate states $WR^\infty$. If the stream is finite, it is terminated with a final value of the computation. The monad $RW^\infty$ is an instance of the resumption monad that in general is not given by a final coalgebra. (Compare also Ahman and Uustalu’s update monads [8].)
Now, assume that $M$ is completely iterative with respect to a module $\overline{M}$. Note that an ‘ordinary’ monad is always a cim with respect to the module $C_0$ (the constant functor returning the initial object), so this assumption does not narrow down the choice of $M$. We prove $MS^\infty$ to be a cim with respect to the module $\overline{M}S^\infty + MSS^\infty$. This means that each coinductive step can unfold the structure ‘horizontally’ (by unfolding more structure of $M$), ‘vertically’ (unfolding the free structure), or both.

**Definition 3.10** For a monad $T$, a distributive law of a $T$-module $\overline{T}$ over a monad $M$ is a distributive law between monads $\lambda : TM \to MT$ together with a natural transformation $\overline{\lambda} : TM \to MT$ such that the obvious analogues of the diagrams for distributive laws between monads commute (except for the diagram for $\eta^T$, since in general $\overline{T}$ is not a monad). Moreover, if $T$ is idealised with $\overline{T}$, we say that the tuple $\langle \lambda, \overline{\lambda} \rangle$ preserves modules if $M\sigma^T \cdot \overline{\lambda} = \lambda \cdot \sigma^T M : TM \to MT$.

**Lemma 3.11** Let $T$ be an idealised monad and let $\langle \lambda, \overline{\lambda} \rangle$ be a module-preserving distributive law of $\overline{T}$ over $M$. Then, the induced monad $MT$ is idealised with $MT$.

One can show that $\lambda : S^\infty M \to MS^\infty$ from the proof of Theorem 3.8 can be extended to a module-preserving distributive law of $SS^\infty$ over $M$. Hence, Lemma 3.11 together with Example 2.4 lead us to the following corollary:

**Theorem 3.12** The monad $MS^\infty$ is idealised with respect to $M\overline{S}^\infty + MSS^\infty$.

We describe a solution in $MS^\infty$ as a two-step process. Intuitively, given an equation morphism $e : X \to MS^\infty(X + A)$, we first unfold ‘horizontally’ via the completely iterative structure of $M$. We are left with what can be seen as an equation morphism in a monad induced by the distributive law $\lambda$ on the Kleisli category of $M$. This morphism (the ‘vertical’ unfolding) has a unique solution, too, which is the desired solution in $MS^\infty$.

**Lemma 3.13** Let $M$ be a cim. Let $e : X \to M(X + A + B)$ factor as follows:

$$X \to \overline{M}(X + A + B) + MA + B \xrightarrow{[\sigma, M(inl-inr), \eta-inr]} M(X + A + B)$$

The morphism $e$ has a unique solution $e^\dagger : X \to M(A + B)$.

**Lemma 3.14** Let $e : X \to MS^\infty(X + A)$ be a morphism that factors as follows:

$$X \to M(SS^\infty(X + A) + A) \xrightarrow{M[\alpha, \eta-inr]} MS^\infty(X + A)$$

Then, $e$ has a unique solution $e^\dagger$ in $MS^\infty$.

**Proof.** Consider the Kleisli category of $M$, denoted as $\mathcal{K}(M)$. For a morphism $f : A \to MB$ in $\mathcal{K}(M)$, we define a $B$-morphism $\hat{f} = \overline{\pi_B} \cdot SF : SA \to SB$. We define an endofunctor $G$ on $\mathcal{K}(M)$ given as $GA = SA$ on objects and $G(f : A \to MB) = \eta_B^M \cdot \hat{f} : SA \to MSB$ on morphisms. The distributive law $\lambda$ induces a monad $\langle S^\infty \rangle$ on $\mathcal{K}(M)$ given by $\langle S^\infty \rangle A = S^\infty A$ and $\langle S^\infty \rangle (f : A \to MB) = \lambda_B \cdot S^\infty f : S^\infty A \to MS^\infty B$. One can show that $\langle S^\infty \rangle$ is the free cim generated by $G$ in $\mathcal{K}(M)$. The morphism $e$ is a guarded equation morphism in $\langle S^\infty \rangle$, so it
has a unique solution \( e^\dagger : X \to M \langle S^\infty \rangle A \). One can verify that it is the desired morphism and that it is unique.

Now, we can prove the main theorem:

**Theorem 3.15** The monad \( MS^\infty \) is completely iterative with respect to the module \( MS^\infty + MSS^\infty \).

**Proof.** In this proof, for brevity, we denote \( S^\infty \) as \( T \) and its module \( SS^\infty \) as \( T \). Let \( e : X \to MT(X + A) \) be a guarded equation morphism. This means that it factors as \( X \longrightarrow MT(X + A) + MT(X + A) + A \xrightarrow{[\sigma^M, M\sigma^T, \eta \text{inr}]} MT(X + A) \). Since \( T \) is an ideal monad, there exist isomorphisms \( T(X + A) \cong T(X + A) + X + A \cong X + T(X + A) + A \).

Thus, \( e \) factors as \( X \longrightarrow M(T(X + A) + A) + M\overline{T}(X + A) + A \xrightarrow{[\sigma^M, M(\text{inl-inr}), \eta \text{inr}]} M(X + T(X + A) + A) \), and it is a guarded equation morphism in the monad \( M \) in the sense of Lemma 3.13. Therefore, there exists a unique solution \( e^\dagger : X \to M(T(X + A) + A) \). The morphism \( M[\sigma^T, \eta^T \cdot \text{inr}] \cdot e^\dagger : X \to MT(X + A) \) is a guarded equation morphism in the sense of Lemma 3.14, so it has a unique solution \( (M[\sigma^T, \eta^T \cdot \text{inr}] \cdot e^\dagger)^\dagger : X \to MTA \). One can verify that it is a unique solution of \( e \) in \( MT \). \( \square \)

**Example 3.16** Consider the monad \( D(\Sigma D)^\infty \) from Example 3.9 (a). It is a cim, which gives us a semantics for discrete Markov processes with sequential composition. An equation morphism \( e : X \to D(\Sigma D)^\infty (X + A) \) is a transition system, where \( X \) is the state space. The solution \( e^\dagger : X \to D(\Sigma D)^\infty A \) is a denotational semantics: for an initial state, it gives us the decision tree – the intermediate (internal) states are forgotten. The solution diagram amounts to adequacy.

For an example of the horizontal and vertical complete iterativity, consider the monad \( D'X = D(1 + X) \) for the terminal object 1, which denotes failure. It is a cim with respect to the module that consists of those values of \( D' \) that fail with the probability at least 0.5. Each transition of a process denoted by \( D'(\Sigma D')^\infty \) can either produce an output from the set \( O \) or perform a silent step, but with the probability of failure at least 0.5.

### 4 Universal property and coproduct

A particularly interesting instance of \( MS^\infty \) is \( M(\Sigma M)^\infty \). In this section, we investigate its universal property: for a cim \( N \), a monad morphism \( M \to N \), and an ideal natural transformation \( \Sigma \to N \), there exists a unique coherent monad morphism \( M(\Sigma M)^\infty \to N \). Informally, morphisms that ‘interpret’ every level of the structure of a resumption in another cim uniquely extend to an interpretation of the whole structure. First, we define three ‘injections’:

\[
\text{liftl} = \left(\Sigma^\infty \xrightarrow{\text{emb} \cdot \Sigma^\eta^h} (\Sigma M)^\infty \xrightarrow{\eta^h (\Sigma M)^\infty} M(\Sigma M)^\infty \right)
\]
\[
\text{liftf} = \left(\Sigma \xrightarrow{\text{emb}} \Sigma^\infty \xrightarrow{\text{liftl}} M(\Sigma M)^\infty \right)
\]
\[
\text{liftg} = \left(\Sigma^\infty \xrightarrow{\text{liftf}} M(\Sigma M)^\infty \right)
\]
It is easy to see that \( \text{liftl} \) and \( \text{liftr} \) are monad morphisms (\( \text{liftl} \) is a composition of two monad morphisms). They are also solution-preserving, the former via \( M(\Sigma M)^\infty \), the latter via \( \overline{M}(\Sigma M)^\infty \).

**Definition 4.1** By \( \Sigma/\text{Cim} \) we denote the following category: objects are ideal natural transformations \( (\Sigma \xrightarrow{f} T) \), where \( T \) is a cim; morphisms are (not necessarily solution-preserving) monad morphisms \( k : T \to N \) such that the following diagram commutes:

\[
\begin{array}{ccc}
\Sigma & \xrightarrow{f} & T \\
\downarrow{g} & & \downarrow{k} \\
N & & 
\end{array}
\]

We define a forgetful functor \( U : \Sigma/\text{Cim} \to \text{Mnd} \) as \( U(\Sigma \xrightarrow{f} M) = M \) on objects and \( Uk = k \) on morphisms.

**Definition 4.2** A monad \( M \) is called \( \Sigma \)-resumable if \( (\Sigma M)^\infty \) exists. By \( \Sigma \text{-Res} \) we denote the full subcategory of \( \text{Mnd} \) with \( \Sigma \)-resumable cims as objects. We denote the inclusion functor by \( I : \Sigma \text{-Res} \to \text{Mnd} \).

We establish the universal property in terms of an \( I \)-relative adjunction. For a discussion on relative adjoints, see, for example, Altenkirch et al. [9].

**Theorem 4.3** The functor \( U \) has an \( I \)-relative left adjoint \( F : \Sigma \text{-Res} \to \Sigma/\text{Cim} \) given by \( FM = (\Sigma \xrightarrow{\text{liftf}} M(\Sigma M)^\infty) \) on objects and \( Fm = m * \left( (\Sigma m(\Sigma M)^\infty + \text{id}) \cdot \xi \right) = m * \iota(\text{emb} \cdot \Sigma m) \) on morphisms.

**Proof.** One can show that the morphism \( Fm \) is a monad morphism, so \( F \) is indeed a functor. Let \( M \) be a \( \Sigma \)-resumable monad, \( N \) be a cim, and \( g : \Sigma \to N \) be an ideal natural transformation. We define the isomorphism

\[
[\cdot] : \Sigma/\text{Cim}[FM, (\Sigma \xrightarrow{g} N)] \cong \text{Mnd}[IM, U(\Sigma \xrightarrow{g} N)] : [\cdot]
\]

by

\[
\begin{align*}
k & : (\Sigma \xrightarrow{\text{liftf}} M(\Sigma M)^\infty) \to (\Sigma \xrightarrow{f} N) \\
m & : M \to U(\Sigma \xrightarrow{f} N) \\
[k] & : M \to U(\Sigma \xrightarrow{f} N) \\
[m] & : (\Sigma \xrightarrow{\text{liftf}} M(\Sigma M)^\infty) \to (\Sigma \xrightarrow{f} N) \\
[m] & = \mu^N \cdot (m * \iota(\text{emb} \cdot \Sigma m))
\end{align*}
\]

For \([m]\) to be a well-defined morphism in \( \Sigma/\text{Cim} \), we note that \([m]\) \cdot \text{liftf} = f. Using the properties of \( M(\Sigma M)^\infty \) and free cims, one can verify that \([\cdot]\) is natural in \( M \) and \( g \), and that \([\cdot]\) and \([\cdot]\) are mutual inverses. \( \square \)

An alternative reading of Theorem 4.3 is that \( (\Sigma \xrightarrow{\text{liftf}} M(\Sigma M)^\infty) \) is the free object in \( \Sigma/\text{Cim} \) generated by a \( \Sigma \)-resumable cim \( M \). This means that for a cim \( N \), an ideal natural transformation \( g : \Sigma \to N \), and a monad morphism \( m : M \to N \), there exists a unique monad morphism \( j : M(\Sigma M)^\infty \to N \) such that the following diagram commutes (one can easily see that \( \text{liftr} \) is the unit of the relative adjunction):
Note that if $M = \text{Id}$, the diagram above instantiates to the fact that $\Sigma^\infty$ is indeed the free cim generated by $\Sigma$. More precisely, the identity monad $\text{Id}$ is initial in $\text{Mnd}$ (the unique monad morphism $! : \text{Id} \to N$ is given by the unit of $N$), so the right-hand side of the diagram above becomes trivial, and what is left is exactly the diagram from Theorem 2.6.

Also, for a solution-preserving monad morphism $n : \Sigma^\infty \to N$, the composition $n \cdot \text{emb} : \Sigma \to N$ is an ideal natural transformation. For a solution-preserving monad morphism $m : M \to N$, there exists a unique monad morphism $j : M(\Sigma M)^\infty \to N$ such that $m = j \cdot \text{liftr}$ and $n \cdot \text{emb} = j \cdot \text{liftf} = j \cdot \text{liftr} \cdot \text{emb}$. This means that $\iota(n \cdot \text{emb}) = \iota(j \cdot \text{liftr} \cdot \text{emb})$, hence, by Lemma 2.7, we get $n = j \cdot \text{liftr}$. Therefore, the morphism $j$ uniquely makes the following diagram commute:

$$\Sigma \xrightarrow{\text{liftr}} M(\Sigma M)^\infty \xrightarrow{\text{liftr}} M$$

$$\downarrow j \quad \downarrow \quad \downarrow m$$

$$\Sigma^\infty \xrightarrow{n} M(\Sigma M)^\infty \xrightarrow{\text{liftr}} N$$

The morphism $j = U[m]_{M,n \cdot \text{emb}}$ does not necessarily preserve solutions, even though $\text{liftr}$, $\text{liftr}$, $m$, and $n$ do. Informally, the action of $j$ on the right component of $M(\Sigma M)^\infty$’s module $M(\Sigma M)^\infty + M\Sigma M(\Sigma M)^\infty$ is not guaranteed to take the leading $M$ into $N$. Nevertheless, we can amend the situation if we know that $N$ is a cim with respect to a two-sided module:

**Definition 4.4** A two-sided module of a monad $N$ is its right module $\langle N, \mu^R \rangle$ together with a natural transformation $\overline{\mu}^R : N N \to N$ such that the obvious diagrams mirroring those of Definition 2.1 commute and $\overline{\mu}^R \cdot \overline{\mu}^L N = \overline{\mu}^L \cdot N\overline{\mu}^R : N N N \to N$. Similarly, we adjust the definition of homomorphisms between modules and idealised monads. We denote the category of monads that are completely iterative with respect to two-sided ideals and solution-preserving monad morphisms as $\text{TCim}$.

In the context of the properties studied in this paper, we can switch from $\text{Cim}$ to $\text{TCim}$ at no cost:

**Theorem 4.5** The category $\text{TCim}$ is a full reflective subcategory of $\text{Cim}$. In other words, the obvious embedding functor $U_{\text{TC}} : \text{TCim} \to \text{Cim}$ has a left adjoint $F_{\text{TC}}$.

In practise, this means that for a monad $N$ completely iterative with respect to a right ideal $N$, there exists a two-sided ideal $\overline{N}$ (given by $NN$) and a module homomorphism $\overline{N} \to \overline{N}$ (that is, every equation morphism guarded via $\overline{N}$ is also guarded via $\overline{N}$), such that $N$ is completely iterative with respect to $\overline{N}$. In such a case, since $j$ from the diagram (1) is equal to $\mu^N \cdot (m\ast\mu^N \cdot ((n \cdot \text{emb}) \ast m))$ and the
right-hand side argument of the left-most \( * \) preserves solutions (Theorem 2.6), the morphism \( j \) is solution-preserving too, which can be verified by a simple diagram chase. In general, the module of \( \Sigma^\infty \) is two-sided, but the module of \( M(\Sigma M)^\infty \) is not. Thus, taking the reflection of the diagram (1) in \( TCiM \), we obtain the following corollary, which is a ‘completely iterative’ counterpart of Hyland, Plotkin and Power’s result that \( M(\Sigma M)^* \) is a coproduct of \( \Sigma^* \) and \( M \) in \( MND \) [26]:

Corollary 4.6 For an endofunctor \( \Sigma \) and a monad \( M \) completely iterative with respect to a two-sided ideal, the \( F_{TC} \)-image of \( M(\Sigma M)^\infty \) (that is, \( M(\Sigma M)^\infty \) idealised with \( M(\Sigma M)^\infty (M(\Sigma M)^\infty + M(\Sigma M)^\infty) \)) is the coproduct of \( \Sigma^\infty \) and \( M \) in \( TCiM \).

5 Discussion

5.1 Complete iterativity

The results presented in this paper are in general contributions to the study of completely iterative monads [4]. We give new examples of cims and describe coproducts with free cims. Note that if \( M \) is ideal, then so is \( MS^\infty \), which means that our constructions scale to the category of ideal cims, if one prefers to work in such a setting.

Our results suggest a form of ‘least- vs greatest fixed points’ duality between ordinary monads and cims: free objects are given by \( F^*A = \mu X.FX + A \) and \( F^\infty A = \nu X.FX + A \) respectively, and coproducts with free objects by \( M(\Sigma M)^* \) and \( M(\Sigma M)^\infty \). There are other constructions on monads that involve initial algebras, for example the coproduct of two ideal monads, as shown by Ghani and Uustalu [20]. We conjecture that a similar construction with \( \nu \) in place of \( \mu \) yields the coproduct in the category of ideal cims.

Adámek et al. [5,6] consider also a finitary case: they define an iterative monad (without ‘completely’) as a finitary monad on a locally finitely presentable category, such that all guarded equation morphisms with finitely presentable object of variables have unique solutions. Similarly, there exists a finitary version of complete Elgot algebras (namely, Elgot algebras) together with an analogue of Theorem 3.5. This suggests that the presented constructions should scale to the finitary case, but we have not yet worked out the details.

Elgot’s original results were set in the context of algebraic theories rather than general category theory. As a future direction of research, it would be interesting to look at structures based on resumptions from the point of view of algebraic specification (operations and equations), especially those that could be used in semantics and programming, like the logging monad from Example 3.9 (b).

5.2 Semantics and programming

As suggested by the present authors in a previous paper [39], models in the style of Moggi [34] of effects generating observable behaviour (such as I/O) require a form of complete iterativity, given that programs need not terminate. Thus, by under-
standing the category of cims, one hopes for a better understanding of such effects. For example, if one pursues modularity, the coproduct in MND of two cims is not in general completely iterative. So, a much better notion of the ‘smallest’ amalgam of such effects would be their coproduct in TCIM. This has a practical aspect: the Haskell programming language is equipped with a single ‘all-inclusive’ IO monad, incorporating effects as diverse as textual interaction, file handling, system calls, the foreign function interface, random number generation, and concurrency; one would hope for more fine-grained components that indicate the actual impure effects in use (see Peyton Jones [27] for discussion).

Resumptions-like structures are used as denotations of processes, that is programs that exhibit observable behaviour in the course of execution (see Abramsky [1]). A monadic structure captures the notion of composition, which allows the resumption monad to serve as a basis of a programming calculus in the style of Moggi [34]. For example, Goncharov and Schröder [21] give a simple semantics for asynchronous side-effecting concurrent processes using the monad $MM^\infty$. We hope that the richer structure of $MS^\infty$, can be used to describe more advanced behaviours and synchronisation of processes along the lines of Abramsky’s interaction categories [2].

In pure functional programming, monads are often used as an embedded domain-specific language (EDSL) used to represent computational effects, built from atomic actions using functoriality and monadic structure. Very often such monadic values describe not necessarily terminating (thus, in a sense, coinductive) programs with non-trivial, parallel resource management, like lazy I/O; see, for example, Kiselyov’s iteratees [28]. Such programs are often represented by data structures similar to monadic resumptions, which are in a close relationship with the outer, ‘imperative’ monad, such as Haskell’s IO. The relationship between the two can be formalised by the operations discussed in this paper: IO actions can be lifted to the level of resumptions (liftr), while the whole EDSL program can be executed, that is flattened back into IO, which can be modelled by the universal property.

5.3 Related work

The notion of complete iterativity was introduced by Elgot [16], and later studied by Aczel et al. [4,30]. The properties of the monad $\Sigma^\infty$ were studied also by Moss [35] and Ghani et al. [19]. Milius and Moss used Elgot algebras [7] to describe solutions to recursive program schemes [31]. The composition $M(\Sigma M)^\infty$ has been previously known to be a monad and a cim in the vertical manner (that is, with respect to the module $M\Sigma M(\Sigma M)^\infty$) [21,39]. In contrast to those results, our proofs do not depend on the resumption monad being given by a final coalgebra.

Resumptions were used in semantics of concurrency in many different shapes and under different names. A domain-theoretic approach to resumptions was taken by Milner [32], Plotkin [40], and Papaspyrou [37]. Interaction trees, that is final coalgebras of functors of the shape $P((-) \times O)^I$ on Set, where $P$ is the powerset functor, were extensively used in Abramsky’s interaction categories [2] (the existence of such final coalgebras was assured by employing Aczel’s non-well-founded
set theory [3]).

In programming, different forms of resumptions were independently rediscovered dozens of times, and used for flexible structuring of programs, though usually without much formal treatment. In the Lisp community, resumptions were dubbed ‘engines’ (Haynes and Friedman [25], Dybvig and Hieb [15]) or ‘trampolined’ programs (Ganz, Friedman, and Wand [18]), while in the Haskell libraries they can be found under the name ‘free monad transformer’ (since liftr from Section 4 is a monad morphism natural in $M$, the functor $M \mapsto M(\Sigma M)\infty$ is a monad transformer in the sense of Moggi [33]). Different programming patterns that involve resumptions were discussed by Claessen [14], Kiselyov [28], Harrison [23], and the present authors [38]. Interleaving of data and effects in the algebraic context was also studied by Filinski and Støvring [17], and Atkey et al. [10].

In type theory, similar constructions were used to model interactive programs (Hancock and Setzer [22]), general recursion via guarded corecursion (Capretta [12]), or semantics of imperative languages (Nakata and Uustalu [36]).

Acknowledgements

We would like to thank Stefan Milius for his remarks on an early draft of this paper, and the anonymous reviewers for their detailed comments and helpful suggestions. The work reported here was partially supported by UK EPSRC-funded project Reusability and Dependent Types (EP/G034516/1).

References

Piróg & Gibbons


287


Towards a Quantum Domain Theory: Order-enrichment and Fixpoints in $W^*$-algebras

Mathys Rennela

Institute for Computing and Information Sciences (iCIS) 
Radboud Universiteit Nijmegen 
Nijmegen, The Netherlands

Abstract
We discuss how the theory of operator algebras, also called operator theory, can be applied in quantum computer science. From a computer scientist point of view, we explain some fundamental results of operator theory and their relevance in the context of domain theory. In particular, we consider the category $W^*$ of $W^*$-algebras together with normal sub-unital maps, provide an order-enrichment for this category and exhibit a class of its endofunctors with a canonical fixpoint.

Keywords: $W^*$-algebras, operator theory, domain theory, fixpoint theorem, quantum computation

Introduction
Our aim here is to use the theory of operator algebras to study the differences and similarities between probabilistic and quantum computations, by unveiling their domain-theoretic and topological structure. To our knowledge, the deep connection between the theory of operator algebras and domain theory was not fully exploited before. This might be due to the fact that the theory of operator algebras, mostly unknown to computer scientists, was developed way before the theory of domains.

Our main contribution is a connection between two different communities: the community of theoretical computer scientists, who use domain theory to study program language semantics (and logic), and the community of mathematicians and theoretical physicists, who use a special class of algebras called $W^*$-algebras to study quantum mechanics.

1 Email: mathys.rennela@gmail.com
Our main purpose was to pave the way to a study of quantum programming language semantics, where types are denoted by W*-algebras, terms are denoted by normal completely positive maps, recursive terms and weakest preconditions are denoted by fixed points (via Dcpo⊥-enrichment), and recursive types are denoted by fixed points of endofunctors (via algebraical compactness).

We list here the main points of the paper, that will be detailed and explained later on:

• Positive maps can be ordered by a generalized version of the Löwner order [Low34], considered in the finite-dimensional case by Selinger in [Sel04]: for two positive maps \( f \) and \( g \), \( f \sqsubseteq g \) iff \( (g - f) \) is positive [Definition 2.1]. We also consider completely positive maps in Section 4.

• Hom-sets of normal (positive) sub-unital maps between W*-algebras are directed-complete with this so-called Löwner order [Theorem 2.2].

• The category of \( W^\ast \) of W*-algebras together with normal sub-unital maps is order-enriched [Theorem 2.9].

• The notion of von Neumann functor is introduced to denote the locally continuous endofunctors on \( W^\ast \) which preserve multiplicative maps [Definition 3.1].

• For such functors, we provide a canonical way to construct a fixpoint, by showing that the category \( W^\ast \) is algebraic compact (for the class of von Neumann functors) [Theorem 3.6]. Our proof are in the lines of Smyth-Plotkin [SP82].

1 A short introduction to operator theory

In this section, we will introduce two structures, known as C*-algebras and W*-algebras. We refer the interested reader to [Tak02] for more details.

1.1 C*-algebras

A Banach space is a normed vector space where every Cauchy sequence converges. A Banach algebra is a linear associative algebra \( A \) over the complex numbers \( \mathbb{C} \) with a norm \( \| \cdot \| \) such that its norm \( \| \cdot \| \) is submultiplicative (i.e. \( \forall x, y \in A, \| xy \| \leq \| x \| \| y \| \)) and turns \( A \) into a Banach space. A Banach algebra \( A \) is unital if it has a unit, i.e. if it has an element \( 1 \) such that \( a \cdot 1 = 1 \cdot a = a \) holds for every \( a \in A \) and \( \| 1 \| = 1 \).

A *-algebra is a linear associative algebra \( A \) over \( \mathbb{C} \) with an operation \( (\cdot)^* : A \to A \) such that for all \( x, y \in A \), the following equations holds: \((x^*)^* = x\), \((x + y)^* = (x^* + y^*)\), \((xy)^* = y^*x^*\) and \((\lambda x)^* = \overline{\lambda} x^*\) (\( \lambda \in \mathbb{C} \)).

A C*-algebra is a Banach *-algebra \( A \) such that \( \| x^* x \| = \| x \|^2 \) for all \( x \in A \). This identity is sometimes called the C*-identity, and implies that every element \( x \) of a C*-algebra is such that \( \| x \| = \| x^* \| \).

Consider now a unital C*-algebra \( A \). An element \( x \in A \) is self-adjoint if \( x = x^* \). An element \( x \in A \) is positive if it can be written in the form \( x = y^* y \), where \( y \in A \).

We write \( A_{sa} \leftrightarrow A \) (resp. \( A^+ \leftrightarrow A \)) for the subset of self-adjoint (resp. positive)
elements of $A$.

For every $C^*$-algebra, the subset of positive elements is a convex cone and thus induces a partial order structure on self-adjoint elements, see [Tak02, Definition 6.12]. That is to say, one can define a partial order on self-adjoint elements of a $C^*$-algebra $A$ as follows: $x \leq y$ if and only if $y - x \in A^+$. From now on, we will consider the following kind of maps of $C^*$-algebras. Let $f : A \to B$ be a linear map of $C^*$-algebras:

**P** The map $f$ is **positive** if it preserves positive elements and therefore restricts to a function $A^+ \to B^+$. A positive map $A \to C$ will be called a state on $A$. It should be noted that positive maps of $C^*$-algebras preserve the order on self-adjoint elements.

**M** The map $f$ is **multiplicative** if $\forall x, y \in A, f(xy) = f(x)f(y)$;

**I** The map $f$ is **involutive** if $\forall x \in A, f(x^*) = f(x)^*$;

**U** The map $f$ is **unital** if it preserves the unit;

**sU** The map $f$ is **sub-unital** if the inequality $0 \leq f(1) \leq 1$ holds;

**cP** For every $C^*$-algebra $A$, one can easily define pointwise a $C^*$-algebra $M_n(A)$ from the set of $n$-by-$n$ matrices whose entries are elements of $A$. The map $f$ is **completely positive** if for every $n \in \mathbb{N}$, $M_n(f) : M_n(A) \to M_n(B)$ defined for every matrix $[x_{i,j}]_{i,j \leq n} \in M_n(A)$ by $M_n(f)([x_{i,j}])_{i,j \leq n} = [f(x_{i,j})]_{i,j \leq n}$ is positive.

For every Hilbert space $H$, the Banach space $B(H)$ of bounded linear maps on $H$ is a $C^*$-algebra. The space $C_0(X)$ of complex-valued continuous functions, that vanish at infinity, on a locally compact Hausdorff space $X$ is a common example of commutative $C^*$-algebra.

Self-adjoint and positive elements of $B(H)$ can be defined alternatively through the inner product of $H$, as in the following standard theorem (see [Con07, II.2.12, VIII.3.8]):

**Theorem 1.1** Let $H$ be a Hilbert space and $T \in B(H)$. Then:

(i) $T$ is self-adjoint if and only if $\forall x \in H, \langle Tx|x \rangle \in \mathbb{R}$;

(ii) $T$ is positive if and only if $T$ is self-adjoint and $\forall x \in H, \langle Tx|x \rangle \geq 0$.

### 1.2 $W^*$-algebras

We will denote by $B(H)$ (resp. $Ef(H)$) the collection of all bounded operators (resp. positive bounded operators below the unit) on a Hilbert space $H$. There are several standard topologies that one can define on a collection $B(H)$ (see [Tak02] for an overview).

**Definition 1.2** The operator norm $\|T\|$ is defined for every bounded operator $T$ in $B(H)$ by: $\|T\| = \sup \{\|T(x)\| \mid x \in H, \|x\| \leq 1\}$. The norm topology is the topology induced by the operator norm on $B(H)$. A sequence of bounded operators $(T_n)$ converges to a bounded operator $T$ in this topology if and only if $\|T_n - T\| \to 0$.

The strong operator topology on $B(H)$ is the topology of pointwise convergence
in the norm of $H$: a net of bounded operators $(T_{\lambda})_{\lambda \in \Lambda}$ converges to a bounded operator $T$ in this topology if and only if $\| (T_{\lambda} - T) x \| \to 0$ for each $x \in H$. In that case, $T$ is said to be strongly continuous.

The weak operator topology on $\mathcal{B}(H)$ is the topology of pointwise weak convergence in the norm of $H$: a net of bounded operators $(T_{\lambda})_{\lambda \in \Lambda}$ converges to a bounded operator $T$ in this topology if and only if $\langle (T_{\lambda} - T) x | y \rangle \to 0$ for $x,y \in H$. In that case, $T$ is said to be weakly continuous.

It is known that, for an arbitrary Hilbert space $H$, the weak operator topology on $\mathcal{B}(H)$ is weaker than the strong operator topology on $\mathcal{B}(H)$, which is weaker than the norm topology on $\mathcal{B}(H)$. However, when $H$ is finite-dimensional, the norm topology is first-countable whereas the other topologies are not necessarily first-countable, see [Tak02, Chapter II.2].

The commutant of $A \subset \mathcal{B}(H)$ is the set $A'$ of all bounded operators that commutes with those of $A$: $A' = \{ T \in \mathcal{B}(H) \mid \forall S \in A, TS = ST \}$. The bicommutant of $A$ is the commutant of $A'$ and will be denoted by $A''$.

The following theorem is a fundamental result in operator theory as it remarkably relates a topological property (being closed in two operator topologies) to an algebraic property (being its own bicommutant).

**Theorem 1.3 (von Neumann bicommutant theorem)** Let $A$ be a unital *-subalgebra of $\mathcal{B}(H)$ for some Hilbert space $H$. The following conditions are equivalent:

(i) $A = A''$.
(ii) $A$ is closed in the weak topology of $\mathcal{B}(H)$.
(iii) $A$ is closed in the strong topology of $\mathcal{B}(H)$.

A $W^*$-algebra (or von Neumann algebra) is a $C^*$-algebra which satisfies one (hence all) of the conditions of the von Neumann bicommutant theorem. The collections of bounded operators on Hilbert spaces are the most trivial examples of $W^*$-algebras. The function space $L^\infty(X)$ for some standard measure space $X$ and the space $\ell^\infty(\mathbb{N})$ of bounded sequences are common examples of commutative $W^*$-algebras.

For every $C^*$-algebra $A$, we denote by $A'$ the dual space of $A$, i.e. the set of all linear maps $\phi : A \to \mathbb{C}$. It is known that a $C^*$-algebra $A$ is a $W^*$-algebra if and only if there is a Banach space $A_*$, called pre-dual of $A$, such that $(A_*)' = A$, see [Sak71, Definition 1.1.2].

A positive map $\phi : A \to B$ between two $C^*$-algebras is normal if every increasing net $(x_{\lambda})_{\lambda \in \Lambda}$ in $A^+$ with least upper bound $\bigvee x_{\lambda} \in A^+$ is such that the net $(\phi(x_{\lambda}))_{\lambda \in \Lambda}$ is an increasing net in $B^+$ with least upper bound $\bigvee \phi(x_{\lambda}) = \phi(\bigvee x_{\lambda})$. 292
1.3 Direct sums and tensors of W*-algebras

The direct sum of a family of C*-algebras \( \{ A_i \}_{i \in I} \) is defined as the C*-algebra

\[
\bigoplus_{i \in I} A_i = \left\{ (a_i)_{i \in I} \in \prod_{i \in I} A_i \mid \sup_{i \in I} \|a_i\| < \infty \right\}
\]

where the operations are defined component-wise and with a norm defined by \( \| (a_i)_{i \in I} \|_\infty = \sup \|a_i\| \).

The direct sum of a family of W*-algebras \( \{ A_i \}_{i \in I} \) is the W*-algebra \( \bigoplus_{i \in I} A_i \) defined as the dual of the C*-algebras \( \bigoplus_{i \in I} A_i \ast \), such that \( A_i \) is the dual of \( A_i \ast \), seen as a C*-algebra.

The spatial tensor product \( A \otimes B \) of two W*-algebras \( A \) with universal normal representations \( \pi_A : A \to \mathcal{B}(H) \) and \( \pi_B : B \to \mathcal{B}(K) \) can be defined as the subalgebra of \( \mathcal{B}(H \otimes K) \) generated by the operators \( mn \in \mathcal{B}(H \otimes K) \) where \( (m, n) \in A \times B \).

**Proposition 1.4** For a W*-algebra \( A \), one has the following properties:

- \( A \oplus 0 = A = 0 \oplus A \);
- \( A \overline{\otimes} 0 = 0 = 0 \overline{\otimes} A \);
- \( A \overline{\otimes} C = A = C \overline{\otimes} A \);
- \( A \overline{\otimes} (\bigoplus_{i \in I} B_i) = \bigoplus_{i \in I} (A \overline{\otimes} B_i) \) for every family of W*-algebras \( \{ A_i \}_{i \in I} \).

2 Rediscovering the domain-theoretic structure of W*-algebras

The W*-algebras together with the normal sub-unital maps (or NsU-maps), i.e. positive and Scott-continuous maps, give rise to a category \( \mathbf{W}^* \), which is a subcategory of the category \( \mathbf{C}^* \) of C*-algebras together with positive sub-unital maps.

In this section, after recalling some standard notion of domain theory, we will show that positive sub-unital maps can be ordered in such a way that the category \( \mathbf{W}^* \) will be \( \mathbf{Dcpo}_{1,1} \)-enriched. The decision of considering normal sub-unital maps instead of normal completely positive maps will be discussed in Section 4.1.

2.1 A short introduction to domain theory

A non-empty subset \( \Delta \) of a poset \( P \) is called directed if every pair of elements of \( \Delta \) has an upper bound in \( \Delta \). We denote it by \( \Delta \subseteq_{\text{dir}} P \). A poset \( P \) is a directed-complete partial order (dcpo) if each directed subset has a least upper bound. A function \( \phi : P \to Q \) between two posets \( P \) and \( Q \) is strict if \( \phi(\bot_P) = \bot_Q \), is monotonic if it preserves the order and Scott-continuous if it preserves directed joins. We denote by \( \mathbf{Dcpo}_{\bot} \) (resp. \( \mathbf{Dcpo}_{1,1} \)) the category with dcpos with bottoms as objects and Scott-continuous maps (resp. strict Scott-continuous maps) as morphisms.
2.2 A Löwner order on positive maps

Since positive elements are self-adjoint, one can define the following order on positive maps of C*-algebras.

**Definition 2.1** [Löwner partial order] For positive maps \( f, g : A \rightarrow B \) between C*-algebras \( A \) and \( B \), we define pointwise the following partial order \( \sqsubseteq \), which turns out to be an infinite-dimensional generalization of the Löwner partial order [Low34] for positive maps: \( f \sqsubseteq g \) if and only if \( \forall x \in A^+, f(x) \leq g(x) \) if and only if \( \forall x \in A^+, (g - f)(x) \in B^+ \) (i.e. \( g - f \) is positive).

One might ask if, for arbitrary C*-algebras \( A \) and \( B \), the poset \( (C^*(A, B), \sqsubseteq) \) is directed-complete. The answer turns out to be no, as shown by our following counter-example:

Let us consider the C*-algebra \( C([0, 1]) := \{ f : [0, 1] \rightarrow C \mid f \text{ continuous} \} \).

The hom-set \( C^*(C, C([0, 1])) \) is isomorphic to \( C([0, 1]) \) if one considers the functions \( F : C^*(C, C([0, 1])) \rightarrow C([0, 1]) \) and \( G : C([0, 1]) \rightarrow C^*(C, C([0, 1])) \) respectively defined by \( F(f) = f(1) \) and \( G(g) = \lambda \alpha \in C. \alpha \cdot g \).

We define an increasing chain \( (f_n)_{n \geq 0} \) of \( C([0, 1]) \) define for every \( n \in \mathbb{N} \) by

\[
 f_n(x) = \begin{cases} 
 0 & \text{if } 0 \leq x < \frac{1}{2} \\
 (x - \frac{1}{2})2^{n+1} & \text{if } \frac{1}{2} \leq x \leq \frac{1}{2} + 2^{-(n+1)} \\
 1 & \text{if } \frac{1}{2} + 2^{-(n+1)} < x \leq 1 
\end{cases}
\]

Suppose that there is a least upper bound \( \phi \) in \( C([0, 1]) \) for this chain. Then, \( \phi(x) = 0 \) if \( x < \frac{1}{2} \). Moreover, \( \lim_{n \rightarrow \infty} \left( \frac{1}{2} + 2^{-(n+1)} \right) = \frac{1}{2} \) implies that \( \phi(x) = 1 \) if \( x > \frac{1}{2} \). It follows that \( \phi(x) \in \{0, 1\} \) if \( x \neq \frac{1}{2} \).

By the Intermediate Value Theorem, the continuity of the function \( \phi \) on the interval \([0, 1]\) implies that there is a \( c \in [0, 1] \) such that \( \phi(c) = \frac{1}{2} \). From \( \phi(c) \notin \{0, 1\} \), we obtain that \( c = \frac{1}{2} \). That is to say \( \phi(\frac{1}{2}) = \frac{1}{2} \), which is absurd since \( f_n(\frac{1}{2}) = 0 \) for every \( n \in \mathbb{N} \).

It follows that there is no least upper bound for this chain in \( C([0, 1]) \) and therefore \( C([0, 1]) \) is not chain-complete. However:

**Theorem 2.2** For \( W^* \)-algebras \( A \) and \( B \), the poset \( (W^*(A, B), \sqsubseteq) \) is directed-complete.

The proof of this theorem will be postponed until after the following lemmas.

**Lemma 2.3** ([RD66], Corollary 1) Let \( f \in C^*(A, B) \) and \( x \in A^+ \). Then, \( f(x) \leq \|x\| \cdot 1 \). Therefore, \( \|f(x)\| \leq \|x\| \).

The following result is known in physics as Vigier’s theorem [Vig46]. A weaker version of this theorem can be found in [Sel04]. It is important in this context because it establishes the link between limits in topology and joins in order theory.
Lemma 2.4 Let $H$ be a Hilbert space. Let $(T_\lambda)_{\lambda \in \Lambda}$ be an increasing net of $\text{Ef}(H)$. Then the least upper bound $\bigvee T_\lambda$ exists in $\text{Ef}(H)$ and is the limit of the net $(T_\lambda)_{\lambda \in \Lambda}$ in the strong topology.

Proof. For any operator $U \in \mathcal{B}(H)$, the inner product $\langle Ux|x \rangle$ is real if and only if $U$ is self-adjoint (by Theorem 1.1). Thus, for each $x \in H$, the net $(\langle T_\lambda x|x \rangle)_{\lambda \in \Lambda}$ of real numbers is increasing, bounded by $\|x\|^2$ and thus convergent to a limit $\lim_\lambda \langle T_\lambda x|x \rangle$ since $\mathbb{R}$ is bounded-complete.

By polarization on norms, $\langle T_\lambda x|y \rangle = \frac{1}{4}(\langle T_\lambda(x+y)|(x+y) \rangle - \langle T_\lambda x|x \rangle - \langle T_\lambda y|y \rangle)$ for any $\lambda \in \Lambda$. Then, for all $x, y \in H$, the limit $\lim_\lambda \langle T_\lambda x|y \rangle$ exists and we can define pointwise an operator $T \in \text{Ef}(H)$ by $\langle Tx|y \rangle = \lim_\lambda \langle T_\lambda x|y \rangle$ for $x, y \in H$.

Indeed, $T$ is the limit of the net $(T_\lambda)_{\lambda \in \Lambda}$ in the weak topology, and therefore in the strong topology since a bounded net of positive operators converges strongly whenever it converges weakly (see [Bla06, I.3.2.8]).

Moreover, $T$ is an upper bound for the net $(T_\lambda)_{\lambda \in \Lambda}$ since $T_\lambda \leq T$ for every $\lambda \in \Lambda$. By Theorem 1.1, if there is a self-adjoint operator $S \in \mathcal{B}(H)$ such that $T_\lambda \leq S$ for every $\lambda \in \Lambda$, then $(T_\lambda x|x) \leq (Sx|x)$ for every $\lambda \in \Lambda$. Thus, $\langle Tx|x \rangle = \lim_\lambda \langle T_\lambda x|x \rangle \leq \langle Sx|x \rangle$. Then, $\langle (S-T)x|x \rangle \geq 0$ for every $x \in H$. By Theorem 1.1, $S-T$ positive and thus $T \leq S$. It follows that $T$ is the least upper bound of $(T_\lambda)_{\lambda \in \Lambda}$. 

Corollary 2.5 For every W*-algebra $A$, the poset $[0,1]_A$ is directed-complete.

Proof. Let $A$ be a W*-algebra. By definition, $A$ is a strongly closed subalgebra of $\mathcal{B}(H)$, for some Hilbert space $H$. Then, let $(T_\lambda)_{\lambda \in \Lambda}$ be an increasing net in $[0,1]_A \subseteq \text{Ef}(H)$. By Lemma 2.4, $(T_\lambda)_{\lambda \in \Lambda}$ converges strongly to $\bigvee T_\lambda \in \text{Ef}(H)$. It follows that $\bigvee T_\lambda \in [0,1]_A$ because $[0,1]_A$ is strongly closed. Thus, $[0,1]_A$ is directed-complete.

This corollary constitutes a crucial step in the proof of Theorem 2.2, as it reveals a link between the topological properties and the order-theoretic properties of W*-algebras.

Lemma 2.6 Any positive map $f : A \to B$ between C*-algebras is completely determined and defined by its action on $[0,1]_A$.

Proof. A positive map of C*-algebras $f : A \to B$ restricts by definition to a map $f : A^+ \to B^+$. Since $f$ preserves the order $\leq$ on positive elements, it restricts to $[0,1]_A \to [0,1]_B$.

Let $x \in A^+ \setminus \{0\}$. From $x \leq \|x\|1$, we can see that $\frac{1}{\|x\|}x \in [0,1]_A$ and thus $f(\frac{1}{\|x\|}x) \in [0,1]_B$. Moreover, $f(x) = \|x\|f(\frac{1}{\|x\|}x)$. This statement can be extended to every element in $A$ since each $y \in A$ is a linear combination of four positive elements (see [Bla06, II.3.1.2]), determining $f(y) \in B$.

We can now show that the poset $(\text{W}^*(A,B), \leq)$ is directed-complete for every pair $A$ and $B$ of W*-algebras.

Proof. [Proof of Theorem 2.2] Let $A$ and $B$ be two W*-algebras. By Corollary 2.5, $[0,1]_A$ and $[0,1]_B$ are directed-complete.
We now consider an increasing net \((f_\lambda)_{\lambda \in \Lambda}\) of \text{NsU-maps} from \(A\) to \(B\), increasing in the Löwner order. Then, for every \(x \in A^+\), there is an increasing net \((f_\lambda(x))_{\lambda \in \Lambda}\) bounded by \(\|x\| \cdot 1\) (by Lemma 2.3).

Moreover, for every non-zero element \(x \in A^+\), from the fact that \([0,1]_B\) is directed-complete, we obtain that the increasing net \((f_\lambda(\|x\|))_{\lambda \in \Lambda}\) has a join \(\bigvee f_\lambda(\|x\|)\) in \([0,1]_B\) and thus we can define pointwise the following upper bound \(f : [0,1]_A \to [0,1]_B\) for the increasing net \((f'\lambda)_{\lambda \in \Lambda}\) of \text{NsU-maps} from \([0,1]_A\) to \([0,1]_B\) such that, for every \(\lambda \in \Lambda\), \(f'_\lambda(x) = f_\lambda(\frac{x}{\|x\|}) (x \neq 0)\): \(f(\frac{x}{\|x\|}) = \bigvee f_\lambda(\frac{x}{\|x\|}) (x \in A^+)\).

This upper bound \(f\) is a positive sub-unital map by construction and can be extended to an upper bound \(f : A \to B\) for the increasing net \((f_\lambda)_{\lambda \in \Lambda}\): for every nonzero \(x \in A^+\), the increasing sequence \((f_\lambda(x))_{\lambda \in \Lambda} = (\|x\| f_\lambda(\frac{x}{\|x\|}))_{\lambda \in \Lambda}\) has a join \(\bigvee f_\lambda(x) = \|x\| \bigvee f_\lambda(\frac{x}{\|x\|})\) in \(B^+\) and thus one can define pointwise an upper bound \(f : A \to B\) for \((f_\lambda)_{\lambda \in \Lambda}\) by \(f(x) = \bigvee f_\lambda(x)\) for every \(x \in A^+\).

We now need to prove that the map \(f\) is normal, by exchange of joins.

Let \((x_\gamma)_{\gamma \in \Gamma}\) be an increasing bounded net in \(A^+\) with join \(\bigvee \gamma x_\gamma\). For every \(\gamma' \in \Gamma\), we observe that \(x_{\gamma'} \leq \bigvee \gamma x_\gamma\) and thus, \(f(x_{\gamma'}) \leq f(\bigvee \gamma x_\gamma)\) (recall that \(f\) preserves the order). As seen earlier, since \([0,1]_B\) is directed-complete, the increasing net \((f(x_\gamma))_{\gamma \in \Gamma}\), which is equal by definition to the increasing net \((\bigvee \gamma f(x_\gamma))_{\gamma \in \Gamma}\), has a join in \(B^+\) defined by \(\bigvee \gamma f(x_\gamma) = \bigvee \gamma \in \Gamma, x_{\gamma'} \neq 0 \|x_{\gamma'}\| f(\frac{x_{\gamma'}}{\|x_{\gamma'}\|})\) if there is a \(\gamma'' \in \Gamma\) such that \(x_{\gamma''} \neq 0\) and by \(\bigvee \gamma f(x_\gamma) = 0\) otherwise. It follows that \(\bigvee \gamma f(x_\gamma) \leq f(\bigvee \gamma x_\gamma)\).

We have to prove now that \(f(\bigvee \gamma x_\gamma) \leq f(x_\gamma)\). Since each map \(f_\lambda (\lambda \in \Lambda)\) is normal, we obtain that \(f(\bigvee \gamma x_\gamma) = \bigvee \lambda (f_\lambda (\bigvee \gamma x_\gamma)) = \bigvee \lambda (\bigvee \gamma f(x_\gamma))\). Moreover, for \(\gamma' \in \Gamma\) and \(\lambda' \in \Lambda\), \(f_{\lambda'}(x_{\gamma'}) \leq \bigvee \lambda f_\lambda (x_{\gamma'}) \leq \bigvee \gamma f(x_\gamma)\). Then, \(\bigvee \gamma f(x_\gamma) \leq \bigvee \gamma (\bigvee \lambda f_\lambda (x_\gamma))\) and thus \(\bigvee \gamma (\bigvee \lambda f_\lambda (x_\gamma)) \leq \bigvee \gamma (\bigvee \gamma f(x_\gamma))\). It follows that \(f(\bigvee \gamma x_\gamma) = \bigvee \lambda (\bigvee \gamma f(x_\gamma)) \leq \bigvee \gamma (\bigvee \lambda f_\lambda (x_\gamma)) = \bigvee \gamma f(x_\gamma)\).

Let \(g \in W^*(A,B)\) be an upper bound for the increasing net \((f_\lambda)_{\lambda \in \Lambda}\). For \(\lambda' \in \Lambda\) and \(x \in A^+\), \(f_{\lambda'}(x) \leq g(x)\). Then, \(\forall x \in A^+, f(x) = \bigvee f_\lambda(x) \leq g(x)\), i.e. \(f \subseteq g\). It follows that \(f\) is the join of \((f_\lambda)_{\lambda \in \Lambda}\). 

This theorem generalizes the fact that the effects of a \(W^*\)-algebra \(A\) (i.e. the positive unital maps from \(\mathbb{C}^2\) to \(A\)) form a directed-complete poset [Tak02, III.3.13-16]. Moreover, it turns out that Theorem 2.2 can be slightly generalized to the following theorem, with a similar proof.

**Theorem 2.7** Let \(A\) and \(B\) be two \(C^*\)-algebras.

If \([0,1]_B\) is directed-complete, then the poset \((C^*(A,B), \subseteq)\) is directed-complete.

### 2.3 Dcpo\(_{\bot\top}\)-enrichment for \(W^*\)-algebras

In this section, we will provide a Dcpo\(_{\bot\top}\)-enrichment for the category \(W^*\) and discuss the domain-theoretic properties of \(C^*\)-algebras.

**Definition 2.8** Let \(C\) be a category for which every hom-set is equipped with the structure of a poset. \(C\) is said to be Dcpo\(_{\bot\top}\)-enriched if its hom-sets are dcpos with
bottom and if the composition of homomorphisms is strict and Scott-continuous, i.e. the pre-composition \((- \circ f : \mathcal{C}(B, C) \to \mathcal{C}(A, C))\) and the post-composition \(h \circ (-) : \mathcal{C}(A, B) \to \mathcal{C}(A, C)\) are strict and Scott-continuous for homomorphisms \(f : A \to B\) and \(h : B \to C\).

**Theorem 2.9** The category \(\mathbf{W}^*\) is a \(\text{Dcpo}_{\perp!}\)-enriched category.

**Proof.**

For every pair \((A, B)\) of \(\mathbf{W}^\ast\)-algebras, \(\mathbf{W}^\ast(A, B)\) together with the Löwner order is a dcpo with zero map as bottom, and therefore \(\mathbf{W}^\ast(A, B) \in \text{Dcpo}_{\perp!}\).

In particular, for every \(\mathbf{W}^\ast\)-algebra \(A\), \(\mathbf{W}^\ast(A, A) \in \text{Dcpo}_{\perp!}\). We consider now for every \(\mathbf{W}^\ast\)-algebra \(A\) a map \(I_A : 1_A = \{\bot_A\} \to \mathbf{W}^\ast(A, A)\) such that \(I_A(\bot) \in \mathbf{W}^\ast(A, A)\) is the identity map on \(A\). The map \(I_A\) is clearly strict Scott-continuous for every \(\mathbf{W}^\ast\)-algebra \(A\).

Then, what need to be proved is that, given three \(\mathbf{W}^\ast\)-algebras \(A, B, C\), the composition \(\circ_{A,B,C} : \mathbf{W}^\ast(B, C) \times \mathbf{W}^\ast(A, B) \to \mathbf{W}^\ast(A, C)\) is Scott-continuous (the strictness of the composition can be easily verified).

We now consider a \(\text{NsU}\)-map \(f : A \to B\) and the increasing net \((g_\lambda)_{\lambda \in \Lambda} \in \mathbf{W}^\ast(B, C)\), with join \(\bigvee_\Lambda g_\lambda \in \mathbf{W}^\ast(B, C)\). One can define an upper bound pointwise by \(u(x) = ((\bigvee_\Lambda g_\lambda) \circ f)(x)\) for the increasing net \((g_\lambda \circ f)_{\lambda \in \Lambda} \in \mathbf{W}^\ast(A, C)\). It is easy to check that \(u\) is a join for the increasing net \((g_\lambda \circ f)_{\lambda \in \Lambda}\); for every upper bound \(v \in \mathbf{W}^\ast(A, C)\) of the increasing net \((g_\lambda \circ f)_{\lambda \in \Lambda}\), we have that \(\forall \lambda \in \Lambda, g_\lambda \circ f \subseteq v\), i.e. \(\forall \lambda \in \Lambda, \forall x \in A^+, g_\lambda(f(x)) \leq v(x)\) and thus \(\forall x \in A^+, u(x) = ((\bigvee_\Lambda g_\lambda) \circ f)(x) = (\bigvee_\Lambda g_\lambda)(f(x)) \leq v(x)\), which implies that \(u \subseteq v\). It follows that the pre-composition is Scott-continuous and, similarly, the post-composition is Scott-continuous.

In operator theory, a \(C^\ast\)-algebra is monotone-complete (or monotone-closed) if it is directed-complete for bounded increasing nets of positive elements. The notion of monotone-completeness goes back at least to Dixmier [Dix51] and Kadison [Kad55] but, to our knowledge, it is the first time that the notion of monotone-completeness is explicitly related to the notion of directed-completeness. The interested reader will find in Appendix A a more detailed correspondence between operator theory and order theory.

It is natural to ask if all monotone-complete \(C^\ast\)-algebras are \(\mathbf{W}^\ast\)-algebras. Dixmier proved that every \(\mathbf{W}^\ast\)-algebra is a monotone-complete \(C^\ast\)-algebra and that the converse is not true [Dix51]. For an example of a subclass of monotone-complete \(C^\ast\)-algebras which are not \(\mathbf{W}^\ast\)-algebras, we refer the reader to a recent work by Saitô and Wright [SW12].

### 3 A fixpoint theorem for endofunctors on \(\mathbf{W}^\ast\)-algebras

In this section, we will show that it is possible to exhibit a fixpoint for a specific class of endofunctors on \(\mathbf{W}^\ast\)-algebras, that we will define later.

For this purpose, we first observe that the one-element \(\mathbf{W}^\ast\)-algebra \(0 = \{0\}\) is the zero object, i.e. initial and terminal object, of the category \(\mathbf{W}^\ast\): a \(\text{NsU}\)-map
f : A \to 0 must be defined by f(x) = 0; a NsU-map g : 0 \to A is linear and thus 
g(0) = 0_A must holds.

We will now consider the following class of functors and then show that they 
amit a canonical fixpoint.

**Definition 3.1** A von Neumann functor is a locally continuous endofunctor on \( W^* \) 
which preserves multiplicative maps.

**Example 3.2** The identity functor and the constant functors on \( W^* \) are locally 
continuous and so does any (co)product of locally continuous functors. It is also 
clear that all those functors preserve multiplicative maps.

Secondly, our proofs and structures will use the notion of embedding-projection 
pairs, that we will define as follows.

**Definition 3.3** An embedding-projection pair is a pair of arrows 
\( \langle e, p \rangle \in C(X, Y) \times C(Y, X) \) in a \( \mathbf{Dcopo}_{\perp} \)-enriched 
category \( C \) such that \( p \circ e = \text{id}_X \) and \( e \circ p \leq \text{id}_Y \).

For two pairs \( \langle e_1, p_1 \rangle, \langle e_2, p_2 \rangle \), it can be shown that \( e_1 \leq e_2 \) iff \( p_2 \leq p_1 \), 
which means that one component of the pair can uniquely determine the other one. We 
denote by \( e^P \) the projection corresponding to a given embedding \( e \), and \( p^E \) the 
embedding corresponding to a given projection \( p \). It should be noted that \((e \circ f)^P = f^P \circ e^P, \quad (p \circ q)^E = q^E \circ p^E \) 
and \(\text{id}^P = \text{id}^E = \text{id} \).

The category \( C^E \) of embeddings of a \( \mathbf{Dcopo}_{\perp} \)-enriched category \( C \) is the subcate-
gory of \( C \) that has objects of \( C \) has objects and embeddings as arrows. It should 
be noted that this category is itself a \( \mathbf{Dcopo}_{\perp} \)-enriched category. Dually, one can 
define the category \( C^P = (C^E)^{op} \) of projections of a \( \mathbf{Dcopo}_{\perp} \)-enriched category \( C \).

An endofunctor \( F \) on a \( \mathbf{Dcopo}_{\perp} \)-enriched category \( C \) is locally continuous (resp. 
locally monotone) if \( F_{X,Y} : C(X, Y) \to C(FX, FY) \) is Scott-continuous (resp. 
monotone).

We can now consider the following setting. Let \( F : W^* \to W^* \) be a von 
Neumann functor. Consider the \( \omega \)-chain \( \Delta = (D_n, \alpha_n)_n \) for which \( D_0 = 0 \), the 
embedding \( \alpha_0 : D_0 \to FD_0 \) is the unique NsU-map from \( D_0 \) to \( FD_0 \), and the 
equalities \( \alpha_{n+1} = F\alpha_n \) and \( D_{n+1} = FD_n \) hold for every \( n \geq 0 \).

Since the endofunctor \( F \) is locally monotone, if for some \( n \in \mathbb{N} \) there is an 
embedding-projection pair \( \langle \alpha_n^E, \alpha_n^P \rangle \), the pair \( \langle \alpha_{n+1}^E, \alpha_{n+1}^P \rangle \) is also 
an embedding-projection pair. It follows that the \( \omega \)-chain \( \Delta \) is well-defined.

**Definition 3.4** Consider the collection \( D = \{ (x_n)_n \in \bigoplus_n D_n \mid \forall n \geq 0, \alpha_n^P(x_{n+1}) = x_n \} \). 
It forms a poset together with the order \( \leq_D \) defined by \( (x_n)_n \leq_D (y_n)_n \equiv 
\forall n \geq 0, x_n \leq_D y_n. \) Moreover, the collection \( D \) can be seen as a \( \ast \)-algebra.

From the fact that the projection \( \alpha_0^P : D_1 \to D_0 = 0 \) is trivially a multiplicative 
map (which maps everything to the unique element of \( 0 \)) and that the functor \( F \) 
preserves multiplicative maps, we can conclude that for every \( n \geq 0 \), the projection 
\( \alpha_{n+1}^P = F\alpha_n^P = \cdots = F^{n+1}\alpha_0^P \) is a NMIsU-map. In fact, it can be shown that 
the embeddings \( \alpha_n : D_n \to D_{n+1} \) are NMIsU-maps as well, by the same reasoning. 
Moreover, it should also be noted that embeddings and projections are strict, i.e.
Considering these facts, one can verify that the collection $D$ forms a $C^*$-algebra with operations defined component-wise on the family of $W^*$-algebras $(D_n)_{n \geq 0}$.

- The unit is defined by $(1_n)_n = (1_{D_n})_n$. From the fact that the embeddings $\alpha_n$ are $N$-maps, i.e. $\alpha_n^P(1) \leq 1$ for every $n \in \mathbb{N}$, we deduce that $1 = (\alpha_n^P \circ \alpha_n^E)(1) \leq \alpha_n^E(1)$ for every $n \in \mathbb{N}$ (recall that the projection $\alpha_n^P$ is an order-preserving map). Hence, every projection $\alpha_n^P$ is a unital map and thus $\alpha_n^P(1_{n+1}) = \alpha_n^P(1_n)$ holds for every $n \in \mathbb{N}$.

- The addition is defined by $(x_n)_n + (y_n)_n = (x_n + y_n)_n$ for all $(x_n)_n, (y_n)_n \in D$. This operation is well-defined: since the projections $\alpha_n^P$ are linear maps, one can observe that $\alpha_n^P(0_{n+1}) = 0_n$ and $\alpha_n^P(x_{n+1} + y_{n+1}) = \alpha_n^P(x_{n+1}) + \alpha_n^P(y_{n+1}) = x_n + y_n$ for every $n \in \mathbb{N}$. Moreover, by the triangle inequality, $\sup_n \|x_n + y_n\| \leq \sup_n \|x_n\| + \sup_n \|y_n\| < \infty$.

- The scalar multiplication is defined by $\lambda(x_n)_n = (\lambda x_n)_n$ for every $\lambda \in \mathbb{C}$ and every $(x_n)_n \in D$. It is easy to verify that this operation is well-defined: $\alpha_n^P(\lambda x_{n+1}) = \lambda \alpha_n^P(x_{n+1}) = \lambda x_n$ for every $n \in \mathbb{N}$ (by linearity of $\alpha_n^P$) and $\|\lambda(x_n)_n\|_\infty = \lambda \|x_n\|_\infty < \infty$.

- The multiplication is defined by $(x_n)_n \cdot (y_n)_n = (x_n \cdot y_n)_n$ for all $(x_n)_n, (y_n)_n \in D$. This operation is well-defined since the projections $\alpha_n^P$ are multiplicative: $\alpha_n^P(x_{n+1} \cdot y_{n+1}) = \alpha_n^P(x_{n+1}) \cdot \alpha_n^P(y_{n+1}) = x_n \cdot y_n$.

Moreover, the Banach spaces $D_n$ are submultiplicative and thus the following inequality holds: $\sup_n \|x_n \cdot y_n\| \leq (\sup_n \|x_n\|)(\sup_n \|y_n\|) < \infty$.

- The involution is defined by $(x_n)_n^* = (x_n^*)_n$ for every $(x_n)_n \in D$. This operation is well-defined since the projections $\alpha_n^P$ are involutive: $\alpha_n^P(x_{n+1}^*) = \alpha_n^P(x_{n+1})^* = x_n^*$. Moreover, as a direct consequence of the $C^*$-identity of the $C^*$-algebras $D_n$, the following equality holds: $\sup_n \|x_n^*\| = \sup_n \|x_n\| < \infty$.

**Proposition 3.5** The $C^*$-algebra $D$ forms a $C^*$-algebra.

**Proof.** Since every $D_n$ is a $C^*$-algebra, the $C^*$-identity holds for $D$ as well: $\|(x_n)_n\|_\infty = \|(x_n^*_n)_n\|_\infty = \sup_n \|x_n^* x_n\| = \sup_n \|x_n\|^2 = (\|(x_n)_n\|_\infty)^2$.

Consider a Cauchy sequence $((x_{m,n})_m)_n \in D$. It follows that for every $n' \in \mathbb{N}$, the following proposition holds: $\forall \varepsilon > 0, \exists M \in \mathbb{N}, \forall m, m' \geq M, \|x_{m,n'} - x_{m',n'}\| \leq \sup_n \|x_{m,n} - x_{m',n}\| < \varepsilon$.

We are required to prove that the Cauchy sequence $((x_{m,n})_m)_n \in D$ converges, by constructing its limit that we will denote by $(l_n)_n$. We can first deduce that, for every $n \in \mathbb{N}$, the sequence $(x_{m,n})_m \in D_n$ is a Cauchy sequence, which converges to a limit $l_n \in D_n$ since $D_n$ is a $W^*$-algebra (and therefore a Banach space). Then, we obtain a sequence $(l_n)_n \in \prod_{n \in \mathbb{N}} D_n$. Since PsU-maps (and therefore NMsU-maps) are contractive, we can conclude that $(l_n)_n \in D$ by the following arguments.

Let $n \in \mathbb{N}$ and $\varepsilon > 0$. From the fact that the inequality $\|x_{m,n+1} - l_{n+1}\| < \varepsilon$ holds, we deduce that $\|\alpha_n^P(x_{m,n+1}) - \alpha_n^P(l_{n+1})\| = \|\alpha_n^P(x_{m,n+1} - l_{n+1})\| < \varepsilon$, and thus, $l_n = \lim_{m \to \infty} x_{m,n} = \lim_{m \to \infty} \alpha_n^P(x_{m,n+1})$. 


\[ \alpha_n^P(l_{n+1}) \]. Moreover, \( \|(l_n)_\infty = \sup_n \|l_n\| < \infty \) since \( l_n \) is the limit of a Cauchy sequence (recall that every Cauchy sequence is bounded). \( \square \)

This leads us to the following new result.

**Theorem 3.6** The category \( \mathbf{W}^* \) is algebraically compact for the class of von Neumann functors, i.e., every von Neumann functor \( F \) admits a canonical fixpoint and there is an isomorphism between the initial \( F \)-algebra and the inverse of the final \( F \)-coalgebra.

The proof of this theorem involves the following notions.

**Definition 3.7** An \( \omega \)-chain in a category \( \mathbf{C} \) is a sequence of the form \( \Delta = D_0 \xlongrightarrow{\alpha_0} D_1 \xlongrightarrow{\alpha_1} \ldots \)

Given an object \( D \) in a category \( \mathbf{C} \), a cocone \( \mu : \Delta \rightarrow D \) for the \( \omega \)-chain \( \Delta \) is a sequence of arrows \( \mu_n : D_n \rightarrow D \) such that the equality \( \mu_n = \mu_{n+1} \circ \alpha_n \) holds for every \( n \geq 0 \).

A colimit (or colimiting cocone) of the \( \omega \)-chain \( \Delta \) is an initial cocone from \( \Delta \) to \( D \), i.e., it has the following universal property: for every cocone \( \mu' : \Delta \rightarrow D' \), there exists a unique map \( f : D \rightarrow D' \) such that the equality \( f \circ \mu_n = \mu'_n \) holds for every \( n \geq 0 \).

Dually, we will consider \( \omega^{\mathbf{op}} \)-chains \( \Delta^{\mathbf{op}} = D_0 \xlongleftarrow{\beta_0} D_1 \xlongleftarrow{\beta_1} \ldots \) in a category, cones \( \gamma : \Delta^{\mathbf{op}} \leftarrow D \) and limits (or limiting cones) for an \( \omega^{\mathbf{op}} \)-chain \( \Delta^{\mathbf{op}} \).

**Proof.** [Proof of Theorem 3.6] Together with its previously defined order, the \( \mathbf{C}^* \)-algebra \( D \) is monotone-complete, since all the \( \mathbf{W}^* \)-algebras \( D_n \) are monotone-complete. Moreover, a separating set of normal states can be defined for \( D \), on the separating set of normal states of the \( \mathbf{W}^* \)-algebras \( D_n \). We can then conclude by Theorem A.4 that \( D \) forms a \( \mathbf{W}^* \)-algebra and we are now required to prove that it can be turn into a colimit for the diagram \( \Delta \).

We now define a cocone \( \Delta \rightarrow D \) which arrows are embeddings \( \mu_n : D_n \rightarrow D(n \geq 0) \) defined by:

- \( \mu_n(x) = ((\alpha_0^P \circ \cdots \circ \alpha_{n-1}^P)(x), (\alpha_1^P \circ \cdots \circ \alpha_{n-1}^P)(x), \ldots, \alpha_{n-1}^P(x), x, \alpha_n^E(x), (\alpha_{n+1}^E \circ \alpha_n^P)(x), \ldots) \) for every \( x \in D_n \). It is easy to check that for every \( m \in \mathbb{N} \) if we define a sequence \( (y_n)_n = \mu_n(x) \), then the equation \( \alpha_n^P(y_{n+1}) = y_n \) holds for every \( n \in \mathbb{N} \). Moreover, as a positive map, the embedding \( \mu_n \) is contracitive and thus \( \|\mu_n(x)\|_{\infty} \leq \sup_n \|x\| = \|x\|_{\infty} < \infty \).
- \( \mu^E_n((x_n)_n) = x_n \) for every \( (x_n)_n \in D \).

Indeed, it is easy to check that those projections \( \mu^P_n \) are NMIV-maps by construction and that the corresponding embeddings \( \mu^E_n \) are NMIVU-maps by construction.

Then, one can see that \( \mu_n^P(\mu^E_n(x)) = x \) for every \( x \in D_n \) and that \( \mu^E_n(\mu_n^P((x_n)_n)) \leq (x_n)_n \) for every \( (x_n)_n \in D \) since:

1. For every \( m \in \mathbb{N} \) such that \( 0 \leq m < n \), \( (\alpha_m^P \circ \cdots \circ \alpha_{n-1}^P)(x_m) = (\alpha_{n-1}^E \circ \cdots \circ \alpha_m^E)(x_m) = x_m \), which implies that \( x_m = (\alpha_m^P \circ \cdots \circ \alpha_{n-1}^P)(x_m) \) and thus \( (\mu^E_n(\mu_n^P((x_n)_n)) = (\alpha_m^P \circ \cdots \circ \alpha_{n-1}^P)(x_m) \) and thus \( ((\mu_n^P \circ \mu^E_n)(x_n))_m = x_m \) for
From the fact that \( \alpha_n^E \circ \alpha_n^P \leq \text{id}_{D_{n+1}} \) for every \( n \in \mathbb{N} \), we obtain that \( \alpha_n^E(x_n) = \alpha_n^E(\alpha_n^P(x_{n+1})) \leq x_{n+1} \) for every \( n \in \mathbb{N} \) and thus by induction, \( \alpha_m^E \circ \cdots \circ \alpha_n^E(x_n) \leq \cdots \leq x_m \) for every \( m \geq n \). Thus, \((\mu_n \circ \mu_n^P)((x_n)_m) \leq x_m \) for every \( m \geq n \).

Moreover, for every \( n \geq 0 \), we observe that \( \mu_n = \mu_{n+1} \circ \alpha_n \) since \( \alpha_n^P(\mu_{n+1}^P(x_{n+1})) = \alpha_n^P(x_{n+1}) = x_n = \mu_n^P((x_n)_n) \) and thus \( \mu_n^P = \alpha_n^P \circ \mu_{n+1}^P = (\mu_n \circ \alpha_n)^P \).

As stated in [RT92], the fact that \( F \) is locally continuous implies that \( \bigvee_n (\mu_n \circ \mu_n^P) = \text{id}_D \), and thus \( \mu : \Delta \to D \) is a colimiting cocone for \( \Delta \) by [SP82, Theorem 2, Proposition A]. Dually, one can show that \( \mu^P : D \to \Delta^P \) is a limiting cone for \( \Delta^P \), the cone of projections \( D_0 \leftarrow \alpha_0^P = D_1 \leftarrow \alpha_1^P = \cdots \).

Since \( F \) is locally continuous, it is therefore locally monotone. It follows that:

- For every \( n \in \mathbb{N} \), \( \langle F \mu_n, F \mu_n^P \rangle \) is an embedding-projection pair;
- The chain \( \{F \mu_n \circ F \mu_n^P \}_n \) is increasing with join \( \bigvee_n (F \mu_n \circ F \mu_n^P) = \text{id}_{FD} \).

From [SP82, Theorem 2] again, we conclude that \( F \mu : F \Delta \to FD \) is a colimiting cocone (and dually \( F \mu^P : FD \to F \Delta^P \) is a limiting cone). Then, we observe that \( F \Delta \) is obtained by removing the first arrow from \( \Delta \) (recall that \( F \alpha_n = \alpha_{n+1} \)). Finally, the fact that two colimiting cocone with the same vertices are isomorphic implies that \( D \) and \( FD \) share the same limit and the same colimit and that there is an isomorphism \( \phi : D \to FD \), i.e. the functor \( F \) admits a fixpoint. \( \square \)

We will now consider as example the construction of the natural numbers.

**Example 3.8** The functor defined by \( FX = X \oplus \mathbb{C} \) gives the chain of embeddings \( 0 \to \mathbb{C} \to \mathbb{C}^2 \to \mathbb{C}^3 \to \cdots \), where \( \mathbb{C}^n \) is the direct sum of \( n \) copies of \( \mathbb{C} \). The relation \( \alpha_{n+1}^E = \alpha_n^E \oplus \text{id}_\mathbb{C} \) holds for every \( n \in \mathbb{N} \) and thus by induction, \( \alpha_n^E = \alpha_0^E \oplus \text{id}_{\mathbb{C}^n} \). Hence, for every \( n \in \mathbb{N} \), \( \alpha_n^E : \mathbb{C}^n = 0 \oplus \mathbb{C}^n \to \mathbb{C}^{n+1} \) is defined by \( \alpha_n^E(c_1, \ldots, c_n) = (0, c_1, \ldots, c_n) \).

Similarly, for every \( n \in \mathbb{N} \), \( \alpha_n^P = \alpha_0^P \oplus \text{id}_{\mathbb{C}^n} : \mathbb{C}^{n+1} \to \mathbb{C}^n \) is defined by \( \alpha_n^P(c_1, c_2, \ldots, c_n) = (c_2, \ldots, c_n) \) and thus the property \( \alpha_n^P(x_{n+1}) = x_n \) holds for every \( (x_n)_n \in \bigoplus_{i \geq 1} \mathbb{C}^i = \bigoplus_{i \geq 0} \mathbb{C} \).

It follows that \( D = \bigoplus_{i \geq 0} \mathbb{C} = \ell^\infty(\mathbb{N}) \) for this functor. More generally, if one consider a functor \( FX = X \oplus A \) where \( A \) is a W*-algebra, then \( D = \bigoplus_{i \geq 0} A \).

## 4 Streams of qubits

We will now consider the functor \( FX = (X \overline{\otimes} A) \oplus \mathbb{C} \) to represent the construction of a list of unbounded length whose elements are in a W*-algebra \( A \). It should be noted that in this setting, the functor \( FX = (X \overline{\otimes} M_2) \oplus \mathbb{C} \) represent the construction of a list of unbounded length whose elements are qubits.

The functor defined by \( FX = (X \overline{\otimes} A) \oplus \mathbb{C} \) gives the chain of embeddings \( 0 \to \mathbb{C} \to A \oplus \mathbb{C} \to 2 \cdot A \oplus A \oplus \mathbb{C} \to \cdots \) where \( n \cdot A \) denotes the spatial tensor of \( n \)
copies of $A$. Assume that this functor has a canonical fixpoint $D$ (this point will be discussed in the next subsection).

From the relation $\alpha_{n+1}^E = (\alpha_n^E \otimes \text{id}_A) \oplus \text{id}_\mathbb{C}(n \in \mathbb{N})$, we obtain by induction that $\alpha_n^E = (\alpha_0^E \otimes \text{id}_n\cdot A) \oplus \text{id}(n-1)\cdot A \oplus \cdots \oplus \text{id}_A \oplus \text{id}_\mathbb{C}$ for every $n \in \mathbb{N}$. It follows that an embedding $\alpha_n^E : n \cdot A \oplus \cdots \oplus A \oplus \mathbb{C} \to (n+1) \cdot A \oplus \cdots \oplus A \oplus \mathbb{C}$ ($n \in \mathbb{N}$) is defined by $\alpha_n^E(\langle a^n_0, \ldots, a^n_n \rangle, \ldots, \langle a^1_0 \rangle, x) = (\langle 0, a^n_0^{-1}, \ldots, a^n_n^{-1} \rangle, \langle a^n_0^{-1}, \ldots, a^n_n^{-1} \rangle, \ldots, \langle a^1_0 \rangle, x)$.

It is clear that the corresponding projection is $\pi_{2,(n+1)\cdot A \oplus \cdots \oplus A \oplus \mathbb{C}}$ and thus $D = \bigoplus_{i \geq 0} i \cdot A$ (where, by convention, we denote $\mathbb{C}$ by $0 \cdot A$).

**Remark 4.1** It is well known that $\bigotimes_{i \geq 1} A$ is the colimit of the (trivial) diagram $A \to A \otimes A \to \cdots$ in $W^*_{\text{NMIU}}$, the category of $W^*$-algebras together with NMIU-maps. However in our framework, the functor $F = -\otimes A$ is associated to the diagram $0 \to 0 \otimes A = 0 \to 0 \otimes A = 0 \to \cdots$.

### 4.1 Remarks about complete positivity

Unfortunately, the functor $FX = (X \otimes A) \oplus \mathbb{C}$ does not preserve NsU-maps. However, one might consider restricting to NcPsU-maps to consider such functor. There is a Dcopo$_{\perp}$-enrichment for the category $W^*_{\text{cP}}$ of $W^*$-algebras together with NcPsU-maps, investigated independently by Cho [Cho14], who proposed the following variation of the Löwner order:

$$f \sqsubseteq_{\text{cP}} g \text{ if and only if } g - f \text{ is completely positive, i.e. } \forall n \forall x. M_n(f)(x) \leq M_n(g)(x).$$

In fact, the following proposition shows that our domain-theoretic structure do not change if one restricts to completely-positive maps.

**Proposition 4.2** Let $f$ and $g$ be two NsU-maps from a $W^*$-algebra $A$ to a $W^*$-algebra $B$. If $f$ and $g$ are completely positive maps, then the relation $f \sqsubseteq_{\text{cP}} g$ holds if and only if the relation $f \sqsubseteq g$ (Definition 2.1) holds.

**Proof.** If $g - f$ is completely positive with $f$ and $g$ completely positive (i.e. $f \sqsubseteq_{\text{cP}} g$), it is therefore positive and thus it is clear that $f \sqsubseteq_{\text{cP}} g$ implies $f \sqsubseteq g$.

Conversely, we will now show that $f \sqsubseteq g$ implies $f \sqsubseteq_{\text{cP}} g$ when $f$ and $g$ are completely positive maps. It is equivalent to show that if $f$ and $f + g$ are completely positive maps, then $g$ is positive implies that $g$ is completely positive.

By the Hahn-Banach theorem, if we consider $P = W^*(A, B)$ as a normed vector space (defined pointwise) and $cP = W^*_{\text{cP}}(A, B)$ as a linear subspace of $P$ and if we consider an element $z \notin P \setminus \text{span}(cP)$, then there is a (continuous) linear map $\varphi : P \to \mathbb{R}$ with $\varphi(x) = 0$ for every $x \in cP$ and $\varphi(z) = 1$.

We will now apply this fact. If the map $g$ is just positive and not completely positive, we obtain that $\varphi(g) = 1$ and therefore $\varphi(f + g) = \varphi(f) + \varphi(g) = 0 + 1 = 1$ by linearity. But this is absurd since, by assumption, the map $f + g$ is completely positive, and thus $\varphi(f + g) = 0$. It follows that $g$ is completely positive. \qed

Then, it is easy to see that every directed join of completely positive maps is
completely positive map as well (using the fact that \( \mathcal{M}(\bigvee_i f_i) = \bigvee_i \mathcal{M}(f_i) \) for every direct set of completely positive maps \( \{f_i\}_i \)).

Moreover, as shown in [Kor12], the direct sums and the spatial tensor products of W*-algebras can be turned into endofunctors \(-\oplus-: W^*_{cP} \times W^*_{cP} \to W^*_{cP}\) and \(-\otimes-: W^*_{cP} \times W^*_{cP} \to W^*_{cP}\), which are von Neumann functors.

**Concluding remarks**

The theorem 2.2 provides a \textbf{Dcpo}⊥!-enrichment for the category \( W^* \) of W*-algebras with NsU-maps, while the theorem 3.6 gives a canonical fixpoint for every multiplicative map-preservering locally continuous endofunctor on \( W^* \). We believe that these two theorems are encouraging enough to consider further investigations of the semantics of quantum computation, using W*-algebras.

**Acknowledgment**

The author would like to thank Bart Jacobs, Sam Staton, Robert Furber, Bas Westerbaan, and Sander Uijlen for helpful discussions. This research has been financially supported by the European Research Council (ERC) under the QCLS grant (Quantum Computation, Logic & Security).

**References**


In this section, we will provide the following correspondence table between operator theory and order theory, where $A$ and $B$ are C*-algebras.

<table>
<thead>
<tr>
<th>Operator Theory</th>
<th>Order theory</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$ monotone-closed</td>
<td>$[0, 1]_A$ directed-complete</td>
<td>A.2</td>
</tr>
<tr>
<td>$f : A \to B$ NsU-map</td>
<td>$f : [0, 1]_A \to [0, 1]_B$ Scott-continuous PsU-map</td>
<td>A.3</td>
</tr>
<tr>
<td>$A$ W*-algebra</td>
<td>$[0, 1]_A$ dcpo with a separating set of normal states</td>
<td>A.4</td>
</tr>
</tbody>
</table>

In the standard literature [Bla06, Tak02], monotone-closed C*-algebras and normal maps are defined as follows.

**Definition A.1** A C*-algebra $A$ is monotone-closed (or monotone-complete) if every bounded increasing net of positive elements of $A$ has a join in $A^+$. A positive map $\phi : A \to B$ between C*-algebras is normal (or a N-map) if every increasing net $(x_\lambda)_{\lambda \in \Lambda}$ in $A^+$ with a join $\bigvee x_\lambda \in A^+$ is such that the net $(\phi(x_\lambda))_{\lambda \in \Lambda}$ is an increasing net in $B^+$ with join $\bigvee \phi(x_\lambda) = \phi(\bigvee x_\lambda)$.

In the standard definition of the notion of monotone-closedness, the increasing nets are not required to be bounded by the unit, like in the definitions we used in this thesis. We will now show that we can assume that the upper bound is the unit, without loss of generality.

**Proposition A.2** A C*-algebra $A$ is monotone-closed if and only if the poset $([0, 1]_A, \leq)$ is directed-complete.

**Proof.** Let $A$ be a C*-algebra.

If $A$ is monotone-closed, then, by definition every increasing net of positive elements bounded by 1 has a join in $[0, 1]_A$ and therefore, the poset $([0, 1]_A, \leq)$ is directed-complete.

Conversely, suppose that $[0, 1]_A$ is directed-complete. We now consider an increasing net of positive elements $(a_\lambda)_{\lambda \in \Lambda}$ in $A^+$, bounded by a nonzero positive element $b \in A^+$. Then, it restricts to an increasing net $(\frac{a_\lambda}{\|b\|})_{\lambda \in \Lambda}$ in $[0, 1]_A$ since $b \leq \|b\| \cdot 1$. By assumption, the increasing net $(\frac{a_\lambda}{\|b\|})_{\lambda \in \Lambda}$ has a join $\bigvee_\lambda \frac{a_\lambda}{\|b\|} \in [0, 1]_A$ and thus $\|b\| \bigvee_\lambda \frac{a_\lambda}{\|b\|}$ is an upper bound for $(a_\lambda)_{\lambda \in \Lambda}$.

Let $c \in A^+$ be an upper bound for the increasing net $(a_\lambda)_{\lambda \in \Lambda}$ such that $c \leq b$. For every $\lambda' \in \Lambda$, $a_{\lambda'} \leq c \leq b \leq \|b\| \cdot 1$ and thus $\frac{c}{\|b\|}$ is an upper bound for the
increasing net \((\frac{a_\lambda}{\|\|})_{\lambda \in \Lambda}\). It follows that \(\bigvee_{\lambda} \frac{a_\lambda}{\|\|} \leq \frac{c}{\|\|}\) and therefore, \(\|b\| \bigvee_{\lambda} \frac{a_\lambda}{\|\|} \leq c\). Thus, \(\|b\| \bigvee_{\lambda} \frac{a_\lambda}{\|\|}\) is the join of the increasing net \((a_\lambda)_{\lambda \in \Lambda}\) bounded by \(b\) and we can conclude that \(A\) is monotone-closed. \(\square\)

In this thesis, we have chosen to use the standard definition of normal maps. However, one can say that a PsU-map is normal if its restriction \(f : [0,1]_A \to [0,1]_B\) is Scott-continuous.

**Proposition A.3** A PsU-map \(f : A \to B\) between C*-algebras is normal if and only if its restriction \(f : [0,1]_A \to [0,1]_B\) is Scott-continuous.

**Proof.** Let \(f : A \to B\) be a positive map between two C*-algebras \(A\) and \(B\).

If \(f\) is normal, then by definition every increasing net \((x_\lambda)_{\lambda \in \Lambda}\) in \([0,1]_A \subseteq A^+\) with join \(\bigvee x_\lambda \in [0,1]_A\) is such that the net \((f(x_\lambda))_{\lambda \in \Lambda}\) is an increasing net in \([0,1]_B \subseteq B^+\) with join \(\bigvee f(x_\lambda) = f(\bigvee x_\lambda) \in [0,1]_B\). That is to say, the restriction \(f : [0,1]_A \to [0,1]_B\) is Scott-continuous.

Conversely, suppose that the restriction \(f : [0,1]_A \to [0,1]_B\) is Scott-continuous. Let \((x_\lambda)_{\lambda \in \Lambda}\) be an increasing net in \(A^+\) with a nonzero join \(y \in A^+\). Since \(y \leq \|y\|, 1\), it restricts to an increasing net \((\frac{x_\lambda}{\|y\|})_{\lambda \in \Lambda}\) in \([0,1]_A\) with a join \(\frac{y}{\|y\|}\). From the Scott-continuity of \(f : [0,1]_A \to [0,1]_B\), we deduce that the net \((f(\frac{x_\lambda}{\|y\|}))_{\lambda \in \Lambda}\) is an increasing net in \([0,1]_B\) with join \(\bigvee f(\frac{x_\lambda}{\|y\|}) = f(\bigvee \frac{x_\lambda}{\|y\|}) \in [0,1]_B\). It follows that the net \((f(x_\lambda))_{\lambda \in \Lambda}\), which is equal to \((\|y\| f(\frac{x_\lambda}{\|y\|}))_{\lambda \in \Lambda}\) by linearity, is an increasing net in \(B^+\) with an upper bound \(\|y\| \bigvee f(\frac{x_\lambda}{\|y\|}) = f(\|y\| \frac{y}{\|y\|}) = f(y) \in B^+\).

Suppose that \(z \in B^+\) is an upper bound for the increasing net \((f(x_\lambda))_{\lambda \in \Lambda}\). From the fact that \(f(x_\lambda) \leq z\) and therefore \(f(\frac{x_\lambda}{\|y\|}) = \frac{f(x_\lambda)}{\|y\|} \leq \frac{z}{\|y\|}\) for every \(\lambda \in \Lambda\), we obtain that \(f(\frac{x_\lambda}{\|y\|}) \leq \frac{z}{\|y\|}\) and thus \(f(y) \leq z\). It follows that \(f(y)\) is the join of the increasing net \((f(x_\lambda))_{\lambda \in \Lambda}\). Hence, we can conclude that the map \(f\) is normal. \(\square\)

It is known that a C*-algebra \(A\) is a W*-algebra if and only if it is monotone-complete and admits sufficiently many normal states, i.e. the set of normal states of \(A\) separates the points of \(A\), see [Tak02, Theorem 3.16]. By combining this fact and Proposition A.2, one can provide an order-theoretic characterization of W*-algebras, as in the following theorem.

**Theorem A.4** Let \(A\) be a C*-algebra.

Then \(A\) is a W*-algebra if and only if its set of effects \([0,1]_A\) is directed-complete with a separating set of normal states (i.e. \(\forall x \in A, \exists f \in W^*(A, [0,1]_C), f(x) \neq 0\)).

The proof will be postponed until after the following theorem, which can be found in [Tak02], and the following lemma.

**Theorem A.5** Every C*-algebra \(A\) admits a faithful (i.e. injective) representation, i.e. an injective *-homomorphism \(\pi : A \to \mathcal{B}(H)\) for some Hilbert space \(H\). A C*-algebra \(A\) is a W*-algebra if and only if there is a faithful representation \(\pi : A \to \mathcal{B}(H)\), for some Hilbert space \(H\), such that \(\pi(A)\) is a strongly-closed subalgebra of \(\mathcal{B}(H)\).
Lemma A.6 For each $W^*$-algebra $A$, there is an isomorphism $A \simeq \text{span}(\mathcal{NS}(A))'$, where $\mathcal{NS}(A) = W^*(A, [0, 1])$ is the collection of normal states of $A$.

Proof. We now consider the map $\zeta_X : X \to X''$ defined by $\zeta_X(x)(\phi) = \phi(x)$ for $x \in X$ and $\phi \in X'$. Let $A$ be a $W^*$-algebra. We observe that $\zeta_{A_*} : A_* \to A'$ is a “canonical embedding” of $A_*$ into $A'$ and it can be proved that $A_*$ is a linear subspace of $A'$ generated by the normal states of $A$, i.e. $\zeta_{A_*}(A_*) = \text{span}(\mathcal{NS}(A))$, see the proof of [Sak71, Theorem 1.13.2]. Then, we can now consider the induced $\zeta$ that there is an isomorphism $\zeta : A \to \text{span}(\mathcal{NS}(A))$, which turns out to be injective (and thus bijective): for every pair $(x, y) \in A_* \times A_*$ such that $x \neq y$, there is a $f \in \mathcal{NS}(A)$ such that $\zeta_{A_*}(x)(f) = f(x) \neq f(y) = \zeta_{A_*}(y)(f)$, which implies that $\zeta_{A_*}(x) \neq \zeta_{A_*}(y)$.

Then for every $W^*$-algebra, from $A_* \simeq \text{span}(\mathcal{NS}(A))$ for every $W^*$-algebra $A$, we obtain that $A = (A_*)' \simeq \text{span}(\mathcal{NS}(A))'$.

□

Proof. [Proof of Theorem A.4] Let $A$ be a $C^*$-algebra.

Suppose that $A$ is a $W^*$-algebra. Then, by Corollary 2.5, $[0, 1]_A$ is a dcpo and thus $A$ is monotone-complete by Proposition A.2. Moreover, we know by Lemma A.6 that there is an isomorphism $\zeta_A : A \to \text{span}(\mathcal{NS}(A))'$ defined by $\zeta_A(a)(\varphi) = \varphi(a)$ for $a \in A$ and $\varphi \in A'$. Therefore, $\zeta_A$ is injective and thus for every pair $(x, y)$ of distinct elements of $A$, $\zeta_A(x) \neq \zeta_A(y)$, which means that there is a $\varphi \in \mathcal{NS}(A)$ such that $\varphi(x) = \zeta_A(x)(\varphi) \neq \zeta_A(y)(\varphi) = \varphi(y)$. It follows that the set $\mathcal{NS}(A)$ is a separating set for $A$.

Conversely, suppose that $A$ is monotone-closed and admits its normal states as a separating set.

There is a representation $\pi : A \to B(H)$, for some Hilbert space $H$, induced by the normal states on $A$, by the Gelfand-Naimark-Segal (GNS) construction [Tak02, Theorem I.9.14, Definition I.9.15]:

- Every normal state $\omega$ on $A$ induces a representation $\pi_\omega : A \to B(H_\omega)$ such that there is a vector $\xi_\omega$ such that $\omega(x) = \langle \pi_\omega(x)\xi_\omega | \xi_\omega \rangle$ for every $x \in A$.
- We define a Hilbert space $H$, which is the direct sum of the Hilbert spaces $H_\omega$, where $\omega$ is a normal state on $A$.
- The representation $\pi : A \to B(H)$ is defined pointwise for every $x \in A$: $\pi(x)$ is the bounded operator on $H$ defined as the direct sum of the bounded operators $\pi_\omega(x)$ on $H_\omega$, where $\omega$ is a normal state on $A$.

By assumption, the set of normal states of $A$ is a separating set for $A$ and thus, for every pair of distinct elements $x, y$ in $A$, there is a state $\rho$ on $A$ such that $\langle \pi_\rho(x)\xi_\rho | \xi_\rho \rangle = \rho(x) \neq \rho(y) = \langle \pi_\rho(y)\xi_\rho | \xi_\rho \rangle$ and thus $\pi_\rho(x) \neq \pi_\rho(y)$ for some state $\rho$ on $A$. It follows that $\pi(x) \neq \pi(y)$ and hence, the representation $\pi$ is faithful.

Let $\rho$ be a normal state on $A$. Since $A$ is monotone-closed, every directed set $(\rho(x_\lambda))_{\lambda \in \Lambda}$ in $B(H)$ has a join $\bigvee_\lambda \rho(x_\lambda) = \rho(\bigvee_\lambda x_\lambda)$. According to the definition we gave earlier of $\pi_\rho$, this imply that $\pi_\rho(x_\lambda)$ converges weakly to $\bigvee_\lambda \pi_\rho(x_\lambda)$. Since a bounded net of positive operators converges strongly whenever it converges weakly (see [Bla06, I.3.2.8]), it turns out that $\bigvee_\lambda \pi_\rho(x_\lambda)$ is the strong limit of $(\pi_\rho(x_\lambda))_{\lambda \in \Lambda}$ in
\( \mathcal{B}(H_\rho) \). Hence, the strong limit of \( (\pi(x_\lambda))_{\lambda \in \Lambda} \) in \( \mathcal{B}(H) \) exists in \( \mathcal{B}(H) \) and is defined as the direct sum of the strong limit of the nets \( (\pi_\omega(x_\lambda))_{\lambda \in \Lambda} \) where \( \omega \) is a normal state on \( A \). Thus, \( \pi(A) \) is strongly closed in \( \mathcal{B}(H) \) and thus \( A \) is a \( \text{W}^* \)-algebra (by Theorem A.5).

It is important to note that, in one of the very first articles about \( \text{W}^* \)-algebras [Kad55], Kadison defined \( \text{W}^* \)-algebras as monotone-closed \( \text{C}^* \)-algebras which separates the points. However, to our knowledge, this definition never became standard.
Identifying All Preorders on the Subdistribution Monad

Tetsuya Sato

Research Institute for Mathematical Sciences
Kyoto University
Kyoto, Japan

Abstract
The countable valuation monad, the countable distribution monad, and the countable subdistribution monad are often used in the coalgebraic treatment of discrete probabilistic transition systems. We identify preorders on them using a technique based on the preorder $\top$-lifting and elementary facts about preorders on real intervals preserved by convex combinations. We show that there are exactly 15, 5, and 41 preorders on the countable valuation monad, the countable distribution monad, and the countable subdistribution monad respectively. We also give concrete definitions of these preorders. By applying Hughes and Jacobs's construction to some preorder on the countable subdistribution monad, we obtain probabilistic bisimulation between Markov chains ignoring states with deadlocks.

Keywords: coalgebras, preorders, monads, probabilistic transition systems, probabilistic bisimulation

1 Introduction
We completely identify preorders on the countable valuation monad $\mathcal{V}$, the countable distribution monad $\mathcal{D}_{\mathbb{N}}$, and the countable subdistribution monad $\mathcal{D}$ on $\text{Set}$ respectively. We list the main results of this paper:

• There are exactly 15 preorders on the monad $\mathcal{V}$, and they are generated from 4 preorders $\sqsubseteq^0$, $\sqsubseteq^1$, $\sqsubseteq^2$, and $\sqsubseteq^3$ (Section 4).
• There are exactly 5 preorders on the monad $\mathcal{D}_{\mathbb{N}}$, and they are generated from the equality $\text{Eq}_{\mathcal{D}_{\mathbb{N}}}$ and the support-inclusion $\sqsubseteq^s$ (Section 5).
• There are exactly 41 preorders on the monad $\mathcal{D}$, and they are generated from 5 preorders $\sqsubseteq^r$, $\sqsubseteq^s$, $\sqsubseteq^d$, $\sqsubseteq^n$, and $\sqsubseteq^M$ (Section 6).
• To identify preorders on $\mathcal{V}$, it is enough to analyse preorders at the singleton type. To identify preorders on $\mathcal{D}$ and $\mathcal{D}_{\mathbb{N}}$, it is enough to analyse preorders at

1 Email:satoutet@kurims.kyoto-u.ac.jp
the Boolean type.

Our task is identifying the class $\text{Pre}(T)$ of preorders on a monad $T$ ($T = V, D_1, D$). We focus on the component $\sqsubseteq$ of each $\subseteq \in \text{Pre}(T)$ at a set $I$. The component $\sqsubseteq_I$ is a preorder on $TI$ that satisfies congruence and substitutivity. We denote by $\text{CSPre}(T, I)$ the set of such preorders on $TI$. We introduce the mapping $(-)_I : \text{Pre}(T) \rightarrow \text{CSPre}(T, I)$ that extracts components at $I$ from preorders on $T$. We calculate preorders on $T$ from $\text{CSPre}(T, I)$ by the left adjoint $\langle - \rangle_I$ and the right adjoint $[-]_I$ of the mapping $(-)_I$, and we analyse the sandwiching situation $\langle \leq \rangle_I \sqsubseteq \subseteq \sqsubseteq \langle \leq \rangle_I$ for each $\leq \in \text{CSPre}(T, I)$, where $\sqsubseteq$ is the component-wise inclusion order for preorders on $T$.

We identify $\text{Pre}(V), \text{Pre}(D_1), \text{Pre}(D)$ as the following steps:

(i) We identify the sets $\text{CSPre}(V, 1), \text{CSPre}(D_1, 2), \text{CSPre}(D, 1)$. Then, the class $\text{Pre}(V)$ is identified by applying [9, Lemma 7].

(ii) We calculate the mappings $\langle - \rangle_I$ and $[-]_I$ for $(T, I) = (D_1, 2)$ and $(T, I) = (D, 1)$. We then identify $\text{Pre}(D_1)$ by proving $\langle - \rangle^2 = [-]_1$. To finish identifying $\text{Pre}(D)$, we analyse the remaining preorders $\subseteq \in \text{Pre}(D)$ such that $\langle \leq \rangle^1 \not\sqsubseteq \subseteq \not\sqsubseteq \langle \leq \rangle^1$ by using preorders on $D_1$.

In [9], Katsumata and the author developed a method to identify preorders on monads, but it is not applied well to the monads $V, D_1, D$. In this paper, we introduce the following new ideas to identify $\text{Pre}(V), \text{Pre}(D_1), \text{Pre}(D)$: in (i) of the above steps, we use Lemma 1.1 to identify congruent and substitutive preorders on the infinite sets $V, D_1, D$. In (ii), we introduce the left adjoint $\langle - \rangle_I$ of the mapping $(-)_I$, and we use the sandwiching situation $\langle \leq \rangle_I \sqsubseteq \subseteq \sqsubseteq \langle \leq \rangle_I$ to identify $\text{Pre}(D_1)$ and $\text{Pre}(D)$.

This work is motivated by a mathematical interest. The author has not found interesting applications of the main results of this work yet, but at least, we have the following contribution: By applying preorders on $D$ to methods in [6,8,9], we discuss coalgebraic simulations between probabilistic transition systems, and obtain probabilistic bisimulations ignoring states with deadlocks between Markov chains (Section 7). From preorders on the monad $D_1$ (resp. $D$), we obtain all precongruences on each typed language given from probabilistic choice $\sum_{i \in I} p_i(-_i)$ (resp. and deadlocks) and several axioms.

1.1 Background

Preorders on monads are equivalent to pointwise preorder enrichments on their Kleisli categories. A suitable partial order on a monad gives a coalgebraic trace semantics [5] and forward/backward simulations between coalgebras [4]. In the studies [6,8], simulations between coalgebras are given from preorders on coalgebra functors systematically. Many of them involve preorders on monads (e.g. the inclusion order $\mathcal{P}(A \times -)$).

In the study [11], precongruences on a typed language with nondeterminism (or) and a divergent term are determined completely, and they are almost equivalent to
preorders on the composite monad $\mathcal{PL}$ of the powerset monad $\mathcal{P}$ and the monad $\mathcal{L}$ given by $\mathcal{L} = 1 + \text{Id}$ [9]. From this point of view, in other words, our work is seen as the variant of [11] for probabilistic systems.

1.2 Preliminaries

Throughout this paper, we work on the category Set of sets and functions. For a monad $(T, \eta, \mu)$ on Set and a function $f : X \to TY$, the Kleisli Lifting $f^T : TX \to TY$ of $f$ is the composition $f^T = \mu \circ T(f)$.

For each set $X$, we denote by $\top_X$ the trivial relation $X \times X$ on $X$, and denote by $\text{Eq}_X$ the equality/diagonal relation on $X$. We denote by $R^\text{op}$ the opposite relation of $R$.

We will use the complete semiring $([0, \infty], +, \cdot, 0, 1)$ for the countable valuation monad; it has arbitrary summations, and an infinite sum is the least upper bound with respect to the standard order $\leq$ of all finite partial sums [3, Volume A, pp. 124–125, denoted by $\mathcal{R}_+$].

The following lemma is crucial to analyse preorders.

**Lemma 1.1** Let $0 < N < \infty$. If $\preceq$ is a preorder on the interval $[0, N]$ that is preserved by convex combinations; in other words, the preorder $\preceq$ satisfies

$$(p_1 \preceq q_1 \land p_2 \preceq q_2 \land t \in [0, 1]) \implies tp_1 + (1-t)p_2 \preceq t(q_1 + (1-t)q_2)$$

then $p \preceq q$ for some $0 < p < q < N$ implies $r \preceq s$ for each $0 < r < s < N$.

**Proof (sketch).** Let $a_n = p^{n+1}/q^n$ and $b_n = \beta^n p + (1 - \beta^n)N$ for each $n \in \mathbb{N}$, where $\beta = (N - q)/(N - p)$. We prove $a_n \preceq b_n$ ($n \in \mathbb{N}$), $\lim_{n \to \infty} a_n = 0$, and $\lim_{n \to \infty} b_n = N$ from $p \preceq q$ and $0 < p < q < N$. We then prove $r \preceq s$ for each $0 < r < s < N$ by using $a_m < r < s < b_m$ and $a_m \preceq b_m$ for some $m \in \mathbb{N}$. \qed

2 Monads for Probabilistic Branching

We first introduce some notations: the sum $d[U]$ of $d : X \to [0, \infty]$ over $U \subseteq X$ is defined by $\sum_{x \in U} d(x)$. The support of $d : X \to [0, \infty]$ is defined by $\text{supp}(d) = \{ x \in X \mid d(x) \neq 0 \}$. The zero distribution $0$ is defined by $0(x) = 0$. The Dirac distribution $\delta_x$ is defined by $\delta_x(x) = 1$ and $\delta_x(y) = 0$ ($x \neq y$).

Next, we define the three monads $\mathcal{V}$, $\mathcal{D}$, and $\mathcal{D}_{\omega}$ on Set as follows:

**Definition 2.1** • We denote by $(\mathcal{V}, \eta^\mathcal{V}, \mu^\mathcal{V})$ the countable valuation monad that is defined as follows: the functor part $\mathcal{V}$ is defined by for each set $X$, $\mathcal{V} X = \{ d : X \to [0, \infty] \mid \omega \geq |\text{supp}(d)| \}$ and $\mathcal{V} f(d)(y) = \sum_{x \in f^{-1}(y)} d(x)$ for each $f : X \to Y$ and $y \in Y$. The unit and multiplication are defined by $\eta^\mathcal{V}_X(x) = \delta_x$ and $(\mu^\mathcal{V}_X(\xi))(x) = \sum_{d \in \mathcal{V} X} \xi(d) \cdot d(x)$ ($x \in X$).

• The countable subdistribution monad $(\mathcal{D}, \eta^\mathcal{D}, \mu^\mathcal{D})$ is defined as follows: for each set $X$, $\mathcal{D} X = \{ d : X \to [0, 1] \mid d[X] \leq 1 \}$, and the unit and the multiplication are inherited from the countable valuation monad.
• The countable distribution monad \( \mathcal{D}_{\mathbb{Q}, \eta_{\mathbb{Q}}, \mu_{\mathbb{Q}}} \) is defined as follows: for each set \( X \), \( \mathcal{D}_{\mathbb{Q}}X = \{ d : X \to [0, 1] \mid d[X] = 1 \} \), and the unit and the multiplication are inherited from the subdistribution monad.

We remark that the condition \( \omega \geq |\text{supp}(d)| \) is automatically obtained from \( d[X] = 1 \ (d[X] \leq 1) \) in the definitions of the (sub)distribution monad.

The probabilistic branching is characterised coalgebraically by \( \mathcal{D} \):

• A Markov chain is characterised as \( \xi : X \to \mathcal{D}X \).
• A probabilistic transition system is characterised as \( \xi : X \to \mathcal{D}(A \times X) \).
• A Segala automaton [13] is characterised as \( \xi : X \to \mathcal{D}(1 + A \times X) \).

Since \( \mathcal{D}X \cong \mathcal{D}((1 + X)) \), we obtain the notion of deadlocks in the probabilistic branching. For example, a Markov chain \( \xi : X \to \mathcal{D}X \) has a deadlock at a state \( x \in X \) when \( \xi(x)[X] < 1 \). For further examples, see [14].

3 The Class of Preorders on a Monad

We introduce some results of [9], which we use to identify preorders on monads. We fix a monad \( (T, \eta, \mu) \) on \( \text{Set} \). We denote it by \( T \) for simplicity.

We define the congruence and substitutivity of preorders on \( TI \) and preorders on the monad \( T \), the latter of which correspond bijectively to pointwise preorder enrichments of the Kleisli category \( \text{Set}_T \) of \( T \).

**Definition 3.1** Let \( I \) be a set, and let \( \preceq \) be a preorder on \( TI \). (i) We call \( \preceq \) congruent if \( (\forall j \in J. f(j) \preceq g(j)) \implies (\forall x \in T J. f^2(x) \preceq g^2(x)) \) for each set \( J \) and functions \( f, g : J \to TI \). (ii) We call \( \preceq \) substitutive if \( f^2 \) is a monotone function on \( (TI, \preceq) \) for each \( f : I \to TI \).

We write \( (\text{CSPre}(T, I), \subseteq) \) for the set of congruent and substitutive preorders on \( TI \), ordered by inclusions. It is closed under opposites and intersections, and it has the greatest and least preorders \( \top_{TI} \) and \( \text{Eq}_{TI} \) respectively.

**Definition 3.2** ([9, Definition 3]) A preorder \( \preceq \) on a monad \( T \) is an assignment of a preorder \( \preceq_I \) on \( TI \) to each set \( I \) such that (i) each \( \preceq_I \) is congruent, and (ii) for each \( f : J \to TI \), \( f^2 \) is a monotone function from \( (T J, \preceq_J) \) to \( (TI, \preceq_I) \) (we also call this property substitutivity).

For example, the assignment \( \preceq \) that is defined by \( A \subseteq X B \iff A \subseteq B \) is indeed a preorder on the powerset monad \( \mathcal{P} \).

We write \( (\text{Pre}(T), \preceq) \) for the class of preorders on \( T \), ordered by the partial order \( \preceq \) defined by \( \preceq \preceq \preceq' \text{ def} \forall I. \preceq_I \subseteq \preceq'_I \). It is closed under these opposites and intersections, which are defined by \( (\preceq^{op})_X = (\preceq_X)^{op} \) and \( (\lambda \preceq \lambda')_X = \lambda \preceq \lambda' \subseteq X \), and it has the least and greatest preorders: the equality \( \text{Eq}^T \) defined by \( \text{Eq}^T_X = \text{Eq}_{TX} = \text{Eq}_{IX} \) and the trivial preorder \( \top^{T} \) defined by \( \top^{T}_X = \top_{TX} \).

For each preorder \( \preceq \) on \( T \), we call \( \preceq_I \) the evaluation at \( I \) of \( \preceq \).
The evaluation mapping \((-)\): \(\subseteq \mapsto \subseteq_I\) is a monotone mapping from \((\text{Pre}(T), \subseteq)\) to \((\text{CSPre}(T, I), \subseteq)\). It has both the right and left adjoints. The right adjoint \([-]^T\) of the evaluation mapping \((-)_I\) is defined by

\[ x \preceq^I y \iff \forall f: X \to TI, f^I(x) \preceq f^I(y). \]

The mapping \([-]^T\) is monotone, and it preserves opposites and intersections.

**Proposition 3.3 ([9, Theorem 3])** For each \(I\), \((-)_I \dashv [-]^T\) and \([-]^T_I = \text{Id}\).

Hence, the preorder \([\preceq]^I\) on \(T\) is the greatest one whose evaluation at \(I\) equals \(\preceq\) for each \(\preceq \in \text{CSPre}(T, I)\).

The left adjoint \((-)^T\) of the evaluation mapping \((-)_I\) is defined by

\[ (-)^I \defeq \bigcap \{ \subseteq \in \text{Pre}(T) \mid \subseteq_I = \preceq \}. \]

The preorder \((\preceq)^I\) on \(T\) is the least one whose evaluation at \(I\) equals \(\preceq\) for each \(\preceq \in \text{CSPre}(T, I)\) since \text{Pre}(T)\) is closed under intersections, and \((\preceq)^I = \preceq\) holds. By using this, we easily obtain that the mapping \((\preceq)^I\) is monotone, that it preserves opposites, and that the adjunction \((\preceq)^T \dashv (\preceq)_I\) holds.

**Lemma 3.4** Let \(\preceq \in \text{CSPre}(T, I)\). If \([\preceq]^I = (\preceq)^I\) then the preorder \([\preceq]^I\) the unique preorder whose evaluation at \(I\) equals \(\preceq\).

We here introduce the opposite-intersection operators on \(\text{Pre}(T)\) and \(\text{CSPre}(T, I)\). The one on \(\text{CSPre}(T, I)\) is given as follows:

\[ \mathcal{C}_{\text{CSPre}(T, I)}(\mathbb{K}) = \left\{ \bigcap L \cap \left( \bigcap M \right)^\Psi \mid L, M \subseteq \mathbb{K} \right\} \text{ where } \mathbb{K} \subseteq \text{CSPre}(T, I) \]

The opposite-intersection closure operator on \(\text{Pre}(T)\) is given in a similar way as the above (we denote it by \(\mathcal{C}_{\text{Pre}(T)}\)). We often write \(\mathcal{C}_{\varpi}\) for simplicity.

### 3.1 Main Results

**Theorem 3.5** Preorders on \(V\), \(\mathcal{D}_3\), and \(\mathcal{D}\) are identified as follows:

(i) \(\text{Pre}(V) = \mathcal{C}_{\varpi}\{\subseteq^0, \subseteq^1, \subseteq^2, \subseteq^3\} \cong \text{CSPre}(V, 1) \cong 15\) where

\[
\begin{align*}
d_1 \subseteq^0_X d_2 & \overset{\text{def}}{=} \text{supp}(d_1) \subseteq \text{supp}(d_2) \\
d_1 \subseteq^1_X d_2 & \overset{\text{def}}{=} \forall x \in X. (d_1(x) \leq d_2(x)) \\
d_1 \subseteq^2_X d_2 & \overset{\text{def}}{=} \forall x \in X. (d_1(x) = d_2(x) \lor d_2(x) = \infty) \\
d_1 \subseteq^3_X d_2 & \overset{\text{def}}{=} \forall x \in X. (d_1(x) \leq d_2(x) \land (d_1(x) = 0 \implies d_2(x) \in \{\infty, 0\})).
\end{align*}
\]

(ii) \(\text{Pre}(\mathcal{D}_3) = \mathcal{C}_{\varpi}\{\subseteq^s, \text{Eq}^{\mathcal{D}_3}\} \cong \text{CSPre}(\mathcal{D}_3, 2) \cong 5\) where

\[
\begin{align*}
d_1 \subseteq^s_X d_2 & \overset{\text{def}}{=} \text{supp}(d_1) \subseteq \text{supp}(d_2).
\end{align*}
\]
(iii) \( \text{Pre}(D) = C_{\nabla, \phi} \{ \sqcup^r, \sqcup^s, \sqcup^d, \sqcup^m, \sqcup^M \} \cong \text{CSPre}(D, 2) \cong 41 \) where

\[
d_1 \sqcup_X^r d_2 \overset{\text{def}}{=} \forall x \in X. d_1(x) \leq d_2(x), \\
d_1 \sqcup_X^s d_2 \overset{\text{def}}{=} \text{supp}(d_1) \subseteq \text{supp}(d_2), \\
d_1 \sqcup_X^d d_2 \overset{\text{def}}{=} (d_1[X] = 1 \implies d_2[\text{supp}(d_1)] = 1), \\
d_1 \sqcup_X^m d_2 \overset{\text{def}}{=} (d_1[X] = 1 \implies d_2 = d_1), \\
d_1 \sqcup_X^M d_2 \overset{\text{def}}{=} (d_1[X] = 1 \implies (d_2[X] = 1 \wedge \text{supp}(d_1) = \text{supp}(d_2))).
\]

We prove (i), (ii), and (iii) of Theorem 3.5 in Section 4, 5, and 6.

4 Preorders on the Countable Valuation Monad

Preorders on a semiring-valued finite multiset monad are pointwise \cite[Lemma 7 and Theorem 8]{}. The following lemma holds by applying this fact to the monad \( V \) with a slight change of cardinality of supports to countable.

Lemma 4.1 Each \( \sqcup \in \text{Pre}(V) \) satisfies \( d_1 \sqcup_X^s d_2 \iff \forall x \in X. d_1(x) \sqcup_1 d_2(x) \), where \( 1 = \{*\} \). Moreover, \( \text{CSPre}(V, 1) \cong \text{Pre}(V) \).

Hence, it suffices to identify \( \text{CSPre}(V, 1) \) to identify \( \text{Pre}(V) \). We regard \( V \cong [0, \infty] \) by the correspondence of each \( d \in V \) with the value \( d(*) \in [0, \infty] \). For each \( \preceq \in \text{CSPre}(V, 1) \), the substitutivity of \( \preceq \) is equivalent to

\[
(p \preceq q \wedge t \in [0, \infty)) \implies tp \preceq tq,
\]

and the congruence of \( \preceq \) is equivalent to

\[
\forall i \in I. (p_i \preceq q_i \wedge t_i \in [0, \infty)) \implies \sum_{i \in I}^{} p_i t_i \preceq \sum_{i \in I}^{} q_i t_i.
\]

Hence, each \( \preceq \in \text{CSPre}(V, 1) \) is preserved by convex combinations.

We partition the set \( V \times V \cong [0, \infty] \times [0, \infty] \) into

\[
R_0 = \{ (0, q) \mid q \in (0, \infty) \}, R_1 = \{ (p, q) \mid 0 < p < q < \infty \}, \\
R_2 = \{ (0, \infty) \}, R_3 = \{ (p, \infty) \mid p \in (0, \infty) \}, R_4 = R_0^\varphi, \\
R_5 = R_1^\varphi, R_6 = R_2^\varphi, R_7 = R_3^\varphi, \text{ and Eq}_{V_1}.
\]

By using Lemma 1.1, we obtain Lemma 4.2 and 4.3.

Lemma 4.2 Let \( \preceq \in \text{CSPre}(V, 1) \). We obtain the following properties:

(i) \( p \preceq \infty \) for some \( 0 < p < \infty \) if and only if \( r \preceq \infty \) for all \( 0 < r \leq \infty \).

This is equivalent to \( R_3 \cap \preceq \neq \emptyset \implies R_3 \subseteq \preceq \).

(ii) \( 0 \preceq \infty \) if and only if \( r \preceq s \) for all \( 0 \leq r \leq \infty \).

This is equivalent to \( R_2 \cap \preceq \neq \emptyset \implies R_2 \cup R_3 \subseteq \preceq \).

(iii) \( p \preceq q \) for some \( 0 < p < q < \infty \) if and only if \( r \preceq s \) for all \( 0 < r < s \leq \infty \).

This is equivalent to \( R_1 \cap \preceq \neq \emptyset \implies R_1 \cup R_3 \subseteq \preceq \).

(iv) \( 0 \preceq q \) for some \( 0 < q < \infty \) if and only if \( r \preceq s \) for all \( 0 \leq r < s \leq \infty \).

This is equivalent to \( R_0 \cap \preceq \neq \emptyset \implies R_0 \cup R_1 \cup R_2 \cup R_3 \subseteq \preceq \).

Lemma 4.3 Let \( \preceq \in \text{CSPre}(V, 1) \). We obtain \( \preceq = \text{Eq}_{V_1} \cup \bigcup_{i \in I} R_i \) where \( I = \{ i \in \{0, 1, \ldots, 7\} \mid R_i \cap \preceq \neq \emptyset \} \).
We prepare the following congruent substitutive preorders on $\mathcal{V}1$:

- $p \leq^{0} q \iff (p > 0 \implies q > 0)$
- $p \leq^{1} q \iff (p \leq q)$
- $p \leq^{2} q \iff (p = q) \lor (q = \infty)$
- $p \leq^{3} q \iff (p \leq q) \land (p = 0 \implies q \in \{\infty, 0\})$

**Proposition 4.4** We obtain $\text{CSPre}(\mathcal{V}, 1) = C_{\gamma, \varphi}\{\leq^{0}, \leq^{1}, \leq^{2}, \leq^{3}\} \cong 15$.

**Proof (Sketch).** Let $\preceq \in \text{CSPre}(\mathcal{V}, 1)$. We define $R(p_{0}, p_{1}, \ldots, p_{7}) = Eq_{\mathcal{V}1} \cup \{R_{i} | p_{i} = \text{true}\}$. By Lemma 4.3, we obtain $\preceq = R(p_{0}, p_{1}, \ldots, p_{7})$ where $p_{i} \iff R_{i} \cap \preceq \neq \emptyset (i \in \{0, 1, \ldots, 7\})$. From Lemma 4.2 and the transitivity of $\preceq$, the octuple $(p_{0}, p_{1}, \ldots, p_{7})$ should satisfy the following formula:

$$P = (p_{0} \implies p_{1} \land p_{2}) \land (p_{1} \lor p_{2} \implies p_{3}) \land (p_{3} \land p_{7} \implies p_{1} \land p_{5}) \land (p_{2} \land p_{7} \implies p_{0}) \land (p_{3} \land p_{6} \implies p_{4}) \land (p_{4} \implies p_{5} \land p_{6}) \land (p_{5} \lor p_{6} \implies p_{7}).$$

We remark that the last 2 clauses of $P$ are given by applying $\leq^{3}$ to Lemma 4.2. There are exactly 15 satisfying assignments of $P$. We then prove by hand,

$$15 \cong \{R(p_{0}, p_{1}, \ldots, p_{7}) | (p_{0}, p_{1}, \ldots, p_{7}) \text{ satisfies } P\} \subseteq C_{\gamma, \varphi}\{\leq^{0}, \leq^{1}, \leq^{2}, \leq^{3}\}.$$

Since $\text{CSPre}(\mathcal{V}, 1) \subseteq \{R(p_{0}, p_{1}, \ldots, p_{7}) | (p_{0}, p_{1}, \ldots, p_{7}) \text{ satisfies } P\}$ and $\leq^{0}, \leq^{1}, \leq^{2}, \leq^{3} \in \text{CSPre}(\mathcal{V}, 1)$, we conclude this proposition. \qed

**Theorem 4.5 (Theorem 3.5(i))** Let $\sqsubseteq^{i}$ be the pointwise ordering generated from $\preceq^{i} (i \in \{0, 1, 2, 3\})$. We obtain $\text{Pre}(\mathcal{V}) = C_{\gamma, \varphi}\{\sqsubseteq^{0}, \sqsubseteq^{1}, \sqsubseteq^{2}, \sqsubseteq^{3}\} \cong 15$.

**Proof.** It is proved immediately from Lemma 4.1 and Proposition 4.4. \qed

## 5 Preorders on the Distribution Monad

First, we identify $\text{CSPre}(\mathcal{D}_{1}, 2)$ where $2 = \{0, 1\}$. We regard $\mathcal{D}_{1}2 \cong [0, 1]$ by the correspondence of each $d = d(0)\delta_{0} + (1 - d(0))\delta_{1} \in \mathcal{D}_{1}2$ with $d(0) \in [0, 1]$. For each $\preceq \in \text{CSPre}(\mathcal{D}_{1}, 2)$, the substitutivity of $\preceq$ is equivalent to

$$p \preceq q \iff \forall t, u \in [0, 1], ((t - u)p + u \preceq (t - u)q + u),$$

and the congruence of $\preceq$ is equivalent to

$$(\forall i \in I. (p_{i} \preceq q_{i}) \land \sum_{i \in I} t_{i} = 1) \implies \sum_{i \in I} p_{i}t_{i} \preceq \sum_{i \in I} q_{i}t_{i}.$$  

Hence, each $\preceq \in \text{CSPre}(\mathcal{V}, 1)$ is preserved by convex combinations.

We partition the set $\mathcal{D}_{1}2 \times \mathcal{D}_{1}2 \cong [0, 1] \times [0, 1]$ into

$R_{0} = \{(0, 1), (1, 0)\}$, $R_{1} = \{(p, q) | p \in \{0, 1\}, 0 < q < 1\}$,

$R_{2} = \{(p, q) | p, q \in \{0, 1\}, p \neq q\}$, $R_{3} = R_{1}^{op}$, and $Eq_{\mathcal{D}_{1}2}$.

By using Lemma 1.1, we obtain Lemma 5.1 and 5.2.
Lemma 5.1 Let $\preceq \in \text{CSPre}(\mathcal{D}_1, 2)$. We obtain the following properties:

(i) $p \preceq q$ for some $0 < p < q < 1$ if and only if $r \preceq s$ for all $r, s \in (0, 1)$.

This is equivalent to $R_2 \cap \preceq \neq \emptyset \implies R_2 \subseteq \preceq$.

(ii) $0 \preceq q$ for some $0 < q < 1$ if and only if $r \preceq s$ for all $(r, s) \in [0, 1] \times (0, 1)$.

This is equivalent to $R_1 \cap \preceq \neq \emptyset \implies R_1 \cup R_2 \subseteq \preceq$.

(iii) $0 \preceq 1$ if and only if $r \preceq s$ for all $r, s \in [0, 1]$.

This is equivalent to $R_0 \cap \preceq \neq \emptyset \implies R_0 \cup R_1 \cup R_2 \cup R_3 \subseteq \preceq$.

Lemma 5.2 Let $\preceq \in \text{CSPre}(\mathcal{D}_1, 2)$. We obtain $\preceq = \mathcal{E}_{\mathcal{D}_1} \cup \bigcup_{i \in I} R_i$ where $I = \{ i \in \{0, 1, 2, 3\} \mid R_i \cap \preceq \neq \emptyset \}$.

Proposition 5.3. We have the following identification:

$$\text{CSPre}(\mathcal{D}_1, 2) = C_{\cap, \varphi} \{ \preceq^s, \mathcal{E}_{\mathcal{D}_1} \} = \{ \top_{\mathcal{D}_1}, \mathcal{E}_{\mathcal{D}_1}, \preceq^s, \preceq^s \circ \preceq^s, \preceq^s \cap \preceq^s \circ \preceq^s \} \cong 5,$$

where $p \preceq^s q \overset{\text{def}}{=} (p \neq q) \implies (0 < q < 1)$.

Proof (Sketch). Analogous to Lemma 4.4 with $R(p_0, p_1, p_2, p_3) = \mathcal{E}_{\mathcal{D}_1} \cup \bigcup_{i \in I} \{ R_i \mid p_i = \text{true} \}$ and the following formula:

$$P = (p_0 \implies p_1 \land p_2 \land p_3) \land (p_1 \implies p_2) \land (p_1 \land p_3 \implies p_0) \land (p_3 \implies p_2)$$

that is obtained from Lemma 5.1 and 5.2 and the transitivity of $\preceq$. There are exactly 5 satisfying assignments $(p_0, p_1, p_2, p_3)$ of $P$. We then prove by hand,

$$5 \cong \{ R(p_0, p_1, p_2, p_3) \mid (p_0, p_1, p_2, p_3) \text{ satisfies } P \} \subseteq C_{\cap, \varphi} \{ \preceq^s, \mathcal{E}_{\mathcal{D}_1} \}.$$ 

Since $\text{CSPre}(\mathcal{D}_1, 2) \subseteq \{ R(p_0, p_1, p_2, p_3) \mid (p_0, p_1, p_2, p_3) \text{ satisfies } P \}$ and $\preceq^s, \mathcal{E}_{\mathcal{D}_1} \in \text{CSPre}(\mathcal{D}_1, 2)$, we conclude this proposition. \hfill \Box

Next, we calculate the mapping $[-]^2 : \text{CSPre}(\mathcal{D}_1, 2) \to \text{Pre}(\mathcal{D}_1)$. Since it preserves intersections and opposites, and $\text{CSPre}(\mathcal{D}_1, 2) = C_{\cap, \varphi} \{ \preceq^s, \mathcal{E}_{\mathcal{D}_1} \}$, it suffices to identify the preorders $[\mathcal{E}_{\mathcal{D}_1}]^2$ and $[\preceq^s]^2$ (we denote it by $\preceq^s$).

Proposition 5.4 The preorders $[\mathcal{E}_{\mathcal{D}_1}]^2$ and $\preceq^s$ are identified as follows:

(i) $d_1 [\mathcal{E}_{\mathcal{D}_1}]^2 \preceq d_2 \iff d_1 = d_2$.

(ii) $d_1 \preceq^s d_2 \iff \text{supp}(d_1) \subseteq \text{supp}(d_2)$.

Next, we calculate the mapping $(-)^2 : \text{CSPre}(\mathcal{D}_1, 2) \to \text{Pre}(\mathcal{D}_1)$.

Lemma 5.5 Let $\preceq \in \text{CSPre}(\mathcal{D}_1, 2)$ and $\alpha \in [0, 1]$. If $d_1, d_2 \in \mathcal{D}_1 \times X$ satisfy the following condition: for each $y \in X$ such that $d_1(y) > d_2(y)$,

$$(\alpha + (1 - \alpha) \frac{d_2(y)}{d_1(y)}) \delta_0 + (1 - \alpha) \left(1 - \frac{d_2(y)}{d_1(y)}\right) \delta_1 \preceq \frac{d_2(y)}{d_1(y)} \delta_0 + \left(1 - \frac{d_2(y)}{d_1(y)}\right) \delta_1$$

then $(\alpha d_1 + (1 - \alpha) d_2) \preceq \preceq^2 d_2$ holds.
Proposition 5.6 The mapping \((-)^2\) equals the mapping \([-]^2\).

Proof (Sketch). We prove the case \(\preceq = \preceq^s \cap \preceq^{s\oplus}\), and omit the other cases.

(Case: \(\preceq = \preceq^s \cap \preceq^{s\oplus}\)) Suppose \(d_1 \preceq d_2\). By Lemma 5.4, it is equivalent to \(\text{supp}(d_1) = \text{supp}(d_2)\). This implies for each \(y \in X\) such that \(d_1(y) > d_2(y)\),

\[
\left(\frac{d_1(y) + d_2(y)}{2d_1(y)} - \frac{d_1(y) - d_2(y)}{2d_1(y)}\right) \preceq \left(\frac{d_2(y)}{d_1(y)} - \frac{d_1(y) - d_2(y)}{d_1(y)}\right).
\]

By Lemma 5.5 with \(\alpha = 1/2\), we obtain \((d_1 + d_2)/2 \preceq \langle \preceq \rangle^{\frac{2}{X}} d_2\). Similarly, we also have \(d_1 \preceq (d_1 + d_2)/2\). Therefore, \([\preceq]^2 = \langle \preceq \rangle^2\) holds. \(\square\)

Theorem 5.7 (Theorem 3.5(ii)) We obtain the following identification:

\[\text{Pre}(\mathcal{D}_1) = C_{\gamma, w}\{\subseteq^s, \text{Eq}_{\mathcal{D}_1}\} = \{\top_{\mathcal{D}_1}, \text{Eq}_{\mathcal{D}_1}, \subseteq^s, \subseteq^{s\oplus}, \subseteq^s \cap \subseteq^{s\oplus}\} \cong 5.\]

Proof. It is proved from Lemma 3.4, Proposition 5.4, 5.3, and 5.6. \(\square\)

6 Preorders on the Subdistribution Monad

First, we identity \(\text{CSPre}(\mathcal{D}, 1)\). We regard \(\mathcal{D} \cong [0, 1]\) by the correspondence of each \(d \in \mathcal{D}\) with the value \(d(*) \in [0, 1]\). For each \(\preceq \in \text{CSPre}(\mathcal{D}, 1)\), the substitutivity of \(\preceq\) is equivalent to

\[(p \preceq q \land t \in [0, 1]) \implies tp \preceq tq,\]

and the congruence of \(\preceq\) is equivalent to

\[\forall i \in I. (p_i \preceq q_i) \land \sum_{i \in I} t_i \leq 1 \implies \sum_{i \in I} p_i t_i \preceq \sum_{i \in I} q_i t_i.\]

Hence, each \(\preceq \in \text{CSPre}(\mathcal{D}, 1)\) is preserved by convex combinations.

We partition the set \(\mathcal{D}_1 \times \mathcal{D}_1 \cong [0, 1] \times [0, 1]\) into

\(\mathcal{R}_0 = \{(0, 1)\}, \mathcal{R}_1 = \{(0, q) : 0 < q < 1\}, \mathcal{R}_2 = \{(p, 1) : 0 < p < 1\}, \mathcal{R}_3 = \{(p, q) : 0 < p < q < 1\}, \mathcal{R}_4 = \mathcal{R}_0^{\oplus}, \mathcal{R}_5 = \mathcal{R}_1^{\oplus}, \mathcal{R}_6 = \mathcal{R}_2^{\oplus}, \mathcal{R}_7 = \mathcal{R}_3^{\oplus},\) and \(\text{Eq}_{\mathcal{D}_1}\).

By using Lemma 1.1, we obtain Lemma 6.1 and 6.2.

Lemma 6.1 Let \(\preceq \in \text{CSPre}(\mathcal{D}, 1)\). We obtain the following properties:

(i) \(p \preceq q\) for some \(0 < p < q < 1\) if and only if \(r \preceq s\) for all \(0 < r < s < 1\).

This is equivalent to \(\mathcal{R}_3 \cap \preceq \neq \emptyset \implies \mathcal{R}_3 \subseteq \preceq\).

(ii) \(0 \preceq q\) for some \(0 < q < 1\) if and only if \(r \preceq s\) for all \(0 < r < s < 1\).

This is equivalent to \(\mathcal{R}_1 \cap \preceq \neq \emptyset \implies \mathcal{R}_1 \cup \mathcal{R}_3 \subseteq \preceq\).

(iii) \(p \preceq 1\) for some \(0 < p < 1\) if and only if \(r \preceq s\) for all \(0 < r < s \leq 1\).

This is equivalent to \(\mathcal{R}_2 \cap \preceq \neq \emptyset \implies \mathcal{R}_2 \cup \mathcal{R}_3 \subseteq \preceq\).

(iv) \(0 \preceq 1\) if and only if \(r \preceq s\) for all \(0 \leq r < s \leq 1\).

This is equivalent to \(\mathcal{R}_0 \cap \preceq \neq \emptyset \implies \mathcal{R}_0 \cup \mathcal{R}_1 \cup \mathcal{R}_2 \cup \mathcal{R}_3 \subseteq \preceq\).

Lemma 6.2 Let \(\preceq \in \text{CSPre}(\mathcal{D}, 1)\). We obtain \(\preceq = \text{Eq}_{\mathcal{D}_1} \cup \bigcup_{i \in I} \mathcal{R}_i\) where \(I = \{i \in \{0, 1, \ldots, 7\} \mid \mathcal{R}_i \cap \preceq \neq \emptyset\}\).
We prepare the following congruent substitutive preorders on $D1$:

- $p \preceq r q \iff p \leq q$
- $p \preceq s q \iff p > 0 \implies q > 0$
- $p \preceq d q \iff p = 1 \implies q = 1$

The superscripts $r$, $s$, and $d$ stand for real values, supports, and deadlocks of distributions respectively. We let $\preceq^{rd} = \preceq^r \cap \preceq^d$ for simplicity.

**Proposition 6.3** We obtain $\text{CSPre}(D, 1) = C_{\gamma, \psi}\{\preceq^r, \preceq^s, \preceq^d\} \cong 25$.

**Proof (Sketch).** Analogous to Lemma 4.4 with $R(p_0, p_1, \ldots, p_7) = Eq_{D1} \cup \bigcup_{i \in I} \{R_i \mid p_i = \text{true}\}$ and the following formula: $P = P' \land P''$ where

$$
P' = (p_0 \iff (p_1 \land p_2)) \land ((p_1 \lor p_2) \implies p_3),
$$

$$
P'' = (p_4 \iff (p_5 \land p_6)) \land ((p_5 \lor p_6) \implies p_7).
$$

There are 25 satisfying assignments $(p_0, p_1, \ldots, p_7)$ of $P$. We prove by hand,

$$25 \cong \{ R(p_0, p_1, \ldots, p_7) \mid (p_0, p_1, \ldots, p_7) \text{ satisfies } P \} \subseteq C_{\gamma, \psi}\{\preceq^r, \preceq^s, \preceq^d\}. $$

Since $\text{CSPre}(D, 1) \subseteq \{ R(p_0, p_1, \ldots, p_7) \mid (p_0, p_1, \ldots, p_7) \text{ satisfies } P \}$ and $\preceq^r, \preceq^s, \preceq^d \in \text{CSPre}(D, 1)$, we conclude this proposition. 

Next, we calculate the mapping $[-]^1 : \text{CSPre}(D, 1) \to \text{Pre}(D)$. Since it preserves intersections and opposites, and $\text{CSPre}(D, 1) = C_{\gamma, \psi}\{\preceq^r, \preceq^s, \preceq^d\}$ holds, it suffices to identify the preorders $[\preceq^r]^1$, $[\preceq^s]^1$, and $[\preceq^d]^1$ (e.g. $[\preceq^d] = [\preceq^s]^{1 \land}$. Let $[\preceq^r] = [\preceq^r]^1$, $[\preceq^s] = [\preceq^s]^1$, and $[\preceq^d] = [\preceq^d]^1$.

**Proposition 6.4** The preorders $\preceq^r$, $\preceq^s$, and $\preceq^d$ are identified as follows:

(i) $d_1 \preceq_X^r d_2 \iff \forall x \in X.d_1(x) \leq d_2(x)$.

(ii) $d_1 \preceq_X^s d_2 \iff \text{supp}(d_1) \subseteq \text{supp}(d_2)$.

(iii) $d_1 \preceq_X^d d_2 \iff (d_1[X] = 1 \implies d_2[\text{supp}(d_1)] = 1)$.

Next, we calculate the mapping $\langle - \rangle^1 : \text{CSPre}(D, 1) \to \text{Pre}(D)$. Generally speaking, $\langle - \rangle^1 : \text{CSPre}(T, I) \to \text{Pre}(T)$ needs not preserve intersections, but the mapping $\langle - \rangle^1 : \text{CSPre}(D, 1) \to \text{Pre}(D)$ preserves intersections.

**Proposition 6.5** The mapping $\langle - \rangle^1$ satisfies the following:

- The mapping $\langle - \rangle^1$ preserves intersections and opposites.
- $\langle \preceq^r \rangle^1 = \preceq^r$, $\langle \preceq^s \rangle^1 = \preceq^s$, and $\langle \preceq^d \rangle^1 = \preceq^m$ where $\preceq^m$ is defined by

$$d_1 \preceq_X^m d_2 \iff (d_1[X] = 1 \implies d_1 = d_2).$$

By Proposition 6.3 and 6.5, we obtain that the preorder $\langle - \rangle^1$ is identified completely for each $\preceq \in \text{CSPre}(D, 1)$ (e.g. $\langle \preceq^d \cap \preceq^{s_{\text{op}}} \rangle^1 = \preceq^m \cap \preceq^{s_{\text{op}}}$).

The following lemma is crucial to identify the mapping $\langle - \rangle^1$.
Lemma 6.6 Let \( \preceq \in \text{CSPre}(D, 1) \). If \( d_1, d_2 \in DX \) satisfy the condition:

\[
\forall x \in \text{supp}(d_1), \left( \frac{1 + d_1[X]}{2} \preceq \frac{(1 + d_1[X]) \min(d_1, d_2)(x)}{d_1(x)} \right)
\]

then we obtain \( d_1 (\preceq)^1_X \min(d_1, d_2) \).

Here, \( \min(d_1, d_2) \in DX \) is defined by \( \min(d_1, d_2)(x) = \min(d_1(x), d_2(x)) \).

Proof of Proposition 6.5 (Sketch). First, we prove \( \preceq^m \in \text{Pre}(D) \). Since \( \preceq^m \in \preceq^d \), the image of the mapping \((\preceq)^1 \) under \( C_{\cap, \preceq} \{\preceq^r, \preceq^s, \preceq^m\} \) is \( \text{CSPre}(D, 1) \). Next, we prove \( d_1 (\preceq)^1_X \min(d_1, d_2) (\preceq)^1_X d_2 \) for each \( d_1 \preceq^m d_2 \) by applying Lemma 6.6 for each \( \preceq \in C_{\cap, \preceq} \{\preceq^r, \preceq^s, \preceq^m\} \). \( \Box \)

To finish identifying the class \( \text{Pre}(D) \), we search for a preorder \( \preceq \) on \( D \) such that \( (\preceq)^1 \preceq \preceq \preceq^d \) for each \( \preceq \in \text{CSPre}(D, 1) \). Here, \( (\preceq)^1 \) and \( \preceq^d \) are the least and greatest preorder on \( D \) whose evaluations at 1 equal \( \preceq \).

Proposition 6.7 Let \( \preceq \in \text{CSPre}(D, 1) \). If \( \preceq \) is one of \( \preceq^d, \preceq^{dp}, \preceq^d \cap \preceq^{sp}, \) and \( \preceq^{dp} \cap \preceq^s \), a preorder \( \preceq \in \text{Pre}(D) \) such that \( (\preceq)^1 \preceq \preceq \preceq^d \) is determined uniquely as follows:

\[
\preceq = \preceq^d \implies \preceq = \preceq_M, \quad \preceq = \preceq^{dp} \implies \preceq = \preceq^{dp}, \quad \preceq = \preceq^d \cap \preceq^{sp} \implies \preceq = \preceq^{dp} \cap \preceq^s \implies \preceq = \preceq^{dp} \cap \preceq^s
\]

where, the preorder \( \preceq_M \in \text{Pre}(D) \) is defined by

\[
d_1 \preceq_X d_2 \overset{\text{def}}{=} (d_1[X] = 1) \implies (d_2[X] = 1 \land \text{supp}(d_1) = \text{supp}(d_2)).
\]

Otherwise, a preorder \( \preceq \in \text{Pre}(D) \) such that \( (\preceq)^1 \preceq \preceq \preceq^d \) does not exist.

Let \( \tau : D_1 \Rightarrow D \) be the natural transformation defined by \( \tau_X(d) = d \) for each \( d \in D_1 X \). For each \( \preceq \in \text{Pre}(D) \), we define the restriction \( C(\preceq) \) of \( \preceq \) by

\[
C(\preceq)_X = \{ (d_1, d_2) \mid \tau_X(d_1) \preceq_X \tau_X(d_2) \} \subseteq D_1 X \times D_1 X.
\]

The following lemma shows that the restriction \( C(\_\_\_\_) \) is a monotone mapping from \( (\text{Pre}(D), \preceq) \) to \( (\text{Pre}(D_1), \preceq) \) since the monotonicity of \( C \) is obvious.

Lemma 6.8 For each \( \preceq \in \text{Pre}(D) \), \( C(\preceq) \) is indeed a preorder on \( D_1 \).

Lemma 6.9 Let \( \preceq \in \text{CSPre}(D, 1) \) and \( \preceq \in \text{Pre}(D) \) with \( (\preceq)^1 \preceq \preceq \preceq^d \).

(i) \( (d_1[X] < 1 \lor d_2[X] < 1) \implies (d_1 \preceq^d_X, d_2 \iff d_1 \preceq_X d_2 \iff d_1 (\preceq)^1_X d_2) \)

(ii) \( C((\preceq)^1) \preceq C(\preceq) \preceq C([\preceq]^1) \)

Hence, each preorder \( \preceq \in \text{Pre}(D) \) such that \( (\preceq)^1 \preceq \preceq \preceq^d \) is determined by preorders on \( D_1 \) between \( C((\preceq)^1) \) and \( C([\preceq]^1) \) and the preorder \( [\preceq]^1 \).

Proof of Proposition 6.7 (Sketch). In the first 4 cases, \( C((\preceq)^1) = \text{Eq}_{D_1} \) and \( C([\preceq]^1) \in \{\preceq^s, \preceq^{sp}\} \). Thus, \( C(\preceq) = \preceq^s \cap \preceq^{sp} \) by Lemma 6.9 (ii). Hence, the
preorder \( \sqsubseteq \) is determined uniquely by Lemma 6.9 (i). Otherwise, \( \langle \leq \rangle \not\sqsubseteq \not\sqsubseteq \langle \leq \rangle \) contradicts Lemma 6.9 (ii) since \( C(\langle \leq \rangle) = \sqsubseteq \cap \sqsubseteq \).

We have finished identifying \( \text{Pre}(D) \).

**Theorem 6.10 (Theorem 3.5(iii))** The class \( \text{Pre}(D) \) is identified as Table 1 below. Moreover, we obtain \( \text{Pre}(D) = C \cap \sqsubseteq \).

\[
\begin{array}{|c|c|}
\hline
\leq & \in \text{CSPre}(D, 1) \\
\hline
\top_D & \top_D \\
\hline
\text{Eq}_D & \text{Eq}_D \\
\hline
\leq^r & \leq^r \\
\hline
\leq^r \cap \leq^s \cap \leq^a & \leq^r \cap \leq^s \cap \leq^a \\
\hline
\leq^r \cap \leq^a & \leq^r \cap \leq^a \\
\hline
\leq^r \cap \leq^s \cap \leq^a & \leq^r \cap \leq^s \cap \leq^a \\
\hline
\leq^s & \leq^s \\
\hline
\leq^s \cap \leq^a & \leq^s \cap \leq^a \\
\hline
\leq^s \cap \leq^a & \leq^s \cap \leq^a \\
\hline
\leq^d \cap \leq^s & \leq^d \cap \leq^s \\
\hline
\leq^d \cap \leq^a & \leq^d \cap \leq^a \\
\hline
\leq^d \cap \leq^s \cap \leq^a & \leq^d \cap \leq^s \cap \leq^a \\
\hline
\leq^d \cap \leq^s \cap \leq^a & \leq^d \cap \leq^s \cap \leq^a \\
\hline
\leq^d & \leq^d \\
\hline
\leq^d \cap \leq^a & \leq^d \cap \leq^a \\
\hline
\leq^d \cap \leq^a & \leq^d \cap \leq^a \\
\hline
\end{array}
\]

Table 1

The table of \( \text{CSPre}(D, 1) \) and \( \text{Pre}(D) \) (we omit opposite preorders)

**Proof.** It is proved immediately from Proposition 6.3, 6.4, 6.5, and 6.7.

The next lemma tells that \( \text{CSPre}(D, 2) \) is enough to identify \( \text{Pre}(D) \).

**Theorem 6.11** We obtain \( \text{Pre}(D) \simeq \text{CSPre}(D, 2) \).

**Proof (Sketch).** By Lemma 3.3, it suffices to prove \( \sqsubseteq \not\simeq \sqsubseteq' \) whenever \( \sqsubseteq \not\simeq \sqsubseteq' \). By [9, Lemma 3], it suffices to compare \( \sqsubseteq \) and \( \sqsubseteq' \) such that \( \sqsubseteq_1 = \sqsubseteq'_1 \).

### 7 Coalgebraic Simulations between Markov Chains

Simulations between coalgebras are defined coalgebraically by using relational liftings of coalgebra functors. In this section, we focus on simulations between Markov chains (i.e. \( D \)-coalgebras). We now discuss relational liftings of \( D \) that are constructed from preorders on \( D \) by the following two methods.

The first method is given in [6,8]. For a given preorder \( \sqsubseteq \in \text{Pre}(D) \), we construct the relational lifting \( D^{(\sqsubseteq)} \) of \( D \) by

\[
D^{(\sqsubseteq)}(R) = \sqsubseteq_X \circ \{ (D\pi_1(d), D\pi_2(d)) \in Dx \times Dy \mid d \in D(R) \} \circ \sqsubseteq_Y
\]

where \( \pi_1: R \to X \) and \( \pi_2: R \to Y \) are projections from a relation \( R \subseteq X \times Y \).
The second method called the preorder \( \sqsubseteq \)-lifting is defined in [9]. For a given \( \preceq \in \text{CSPre}(D, I) \), we define the relational lifting \( D^\sqsubseteq(R) \) of \( D \) by

\[
D^\sqsubseteq(R) = \{ (f : X \to DI, g : Y \to DI) \mid \forall (x, y) \in R. f(x) \preceq g(y) \}
\]

\[
D^\sqsubseteq\sqsubseteq(R) = \{ (d_1, d_2) \in DX \times DY \mid \forall (f, g) \in D^\sqsubseteq(R). f(d_1) \preceq g(d_2) \}.
\]

We relate earlier studies of (bi)simulations between probabilistic transition systems to the coalgebraic simulations obtained by the above methods:

- \( D^\cap\sqsubseteq\sqsubseteq \)-simulation is equivalent to the generalised definition of probabilistic bisimulation [10] given at [2, Definition 4.2]. At [2, Theorem 4.5], assuming z-closedness [2, Section 2], the generalised definition is equivalent to \( D^{\text{EqDe}} \)-simulation (i.e. \( D \)-bisimulation [2, Section 3]).
- We obtain \( D^\cap\sqsubseteq\sqsubseteq(R) = \{ (d_1, d_2) \mid \forall U \subseteq X. d_1[U] \leq d_2[R(U)] \}. \) Hence, the probabilistic bisimulation defined in [16] is equivalent to \( D^\cap\sqsubseteq\sqsubseteq(-) \cap (D^\cap\sqsubseteq\sqsubseteq(-))^{\text{De}} \)-simulation [12]. Moreover, the relational lifting for probabilistic simulations defined at [1, Definition 3.6] equals \( D^{\sqsubseteq\sqsubseteq} \sqsubseteq\sqsubseteq(R) \).
- Jonsson-Larsen simulations over Markov chains are \( D^{\sqsubseteq\sqsubseteq} \)-simulations [4].
- \( D^{\sqsubseteq\sqsubseteq} \)-simulations, \( D^\cap\sqsubseteq\sqsubseteq \)-simulations, and support simulation that are defined as follows are equivalent: \( R \) is a support (bi)simulation between Markov chains \((X, \xi) \) and \((Y, \xi') \) if \( R \) is a (bi)simulation between two \( P \)-colagebras \((X, \text{supp} \circ \xi) \) and \((Y, \text{supp} \circ \xi') \) in the standard sense.
- \( D^{\sqsubseteq\sqsubseteq\text{De}} \)-simulations and \( D^\cap\sqsubseteq\sqsubseteq \sqsubseteq\sqsubseteq \)-simulations are equivalent to support bisimulations and support bisimulations up to \( \sqsubseteq \)-closedness respectively.

We apply the preorders \( \sqsubseteq^m \), \( \sqsubseteq^M \), and \( \sqsubseteq^d \) on \( D \) to the construction \( D^\sqsubseteq \). For two Markov chains \((X, \xi) \) and \((Y, \xi') \), a binary relation \( R \subseteq X \times Y \) is:

- a \( D^{\sqsubseteq^m} \)-simulation if and only if \( (x, y) \in R \land \xi(x)[X] = 1 \implies (\xi(x), \xi'(y)) \in D^{\text{EqDe}}(R) \).
  This is seen as a probabilistic bisimulation ignoring states with deadlocks.
- a \( D^{\sqsubseteq^M} \)-simulation if and only if \( (x, y) \in R \land \xi(x)[X] = 1 \implies (\xi(x), \xi'(y)) \in D^{\sqsubseteq\sqsubseteq \sqsubseteq\sqsubseteq}(R) \).
  This is seen as a support-bisimulation ignoring states with deadlocks.
- a \( D^{\sqsubseteq^d} \)-simulation if and only if \( (x, y) \in R \land \xi(x)[X] = 1 \implies (\xi(x), \xi'(y)) \in D^{\sqsubseteq\sqsubseteq}(R) \).
  This is seen as a reverse support simulation ignoring states with deadlocks.

320
We give an example of $\mathcal{D}(\mathbb{E}^n)$-simulation. We consider two Markov chains $(X, \xi)$ and $(Y, \xi')$ and their start states $x \in X$ and $y \in Y$ as in the right figure. The dashed arrows are a $\mathcal{D}(\mathbb{E}^n)$-simulation $R$ between $x$ and $y$. First, since the state $x$ has a deadlock, the states $x$ and $y$ are assumed to be probabilistic bisimilar unconditionally. Next, since transitions started from the state $x'$ has no deadlock, the state $y'$ must be probabilistic bisimilar to the state $x'$ in the sense of $\mathcal{D}(\mathbb{E}^n)$-(bi)similarity.

8 Future Work

We have the following future work at this time:

- We expect to analyse preorders on other monads. For example, the convex module monad $\mathcal{CM}$ [7,15] that captures discrete probabilistic branching combined with nondeterminism.
- We expect to obtain preorders on the composite monad $ST$ of monads $S$ and $T$ by using a distributive law $\delta: TS \Rightarrow ST$ from preorders on $S$ and $T$.

Acknowledgement

The author thanks to Shin-ya Katsumata for valuable comments and stimulating discussions, Paul Blain Levy for constructive suggestions, and Naohiko Hoshino, Ichiro Hasuo, and Corina Cîrstea for helpful comments.

References


